4D - Langage

- Texte multistyle
- Transactions
- Triggers
- Utilisateurs et groupes
- Variables
- Web Services (Client)
- Web Services (Serveur)
- XML
- XML DOM
- XML SAX
- Zone Web
- Liste des thèmes de constantes
- Codes d'erreurs
- Codes de caractères
- Liste des nouveautés
- Commandes obsolètes
- Liste alphabétique des commandes

Introduction

Copyrights et mentions légales

Préface

4D possède son propre langage de programmation. Ce langage intégré, qui comprend plus de 1000 commandes, fait de 4D un outil de développement très puissant.

Le langage de 4D peut être utilisé pour de multiples types de tâches, de la réalisation de calculs simples à la création d'interfaces utilisateur personnalisées complexes. Par l'intermédiaire des routines du langage, vous pourrez par exemple :

- Accéder par programmation à tous les éditeurs de gestion des enregistrements (tris, recherches...),
- Créer et imprimer des états et des étiquettes complexes avec les données de la base,
- Communiquer avec d'autres systèmes d'information,
- Gérer des documents,
- Importer et exporter des données entre des bases 4D et d'autres applications,
- Incorporer des procédures écrites dans d'autres langages que celui de 4D.

La flexibilité et la puissance du langage de 4D en font l'outil idéal pour toute une gamme de fonctions de gestion de l'information. Les utilisateurs novices peuvent rapidement effectuer des calculs. Les utilisateurs expérimentés peuvent personnaliser leurs bases sans devoir connaître la programmation. Les développeurs plus expérimentés peuvent utiliser la puissance du langage de 4D pour ajouter à leurs bases de données des fonctions sophistiquées, allant jusqu'au transfert de fichiers et aux communications. Les développeurs maîtrisant la programmation dans d'autres langages peuvent ajouter leurs propres routines au langage de 4D.

Le langage de programmation de 4D s'enrichit lorsqu'un plug-in est installé dans l'application. Chaque plug-in comporte en effet des commandes spécifiques, permettant de gérer les fonctions qu'il accomplit.

A propos des manuels

Les manuels de 4D, listés ci-dessous, s'appliquent indifféremment à décrire le fonctionnement de 4D et de 4D Server. Une seule exception : le "Manuel de référence 4D Server", qui traite uniquement des fonctions de 4D Server.

- Le manuel "Langage" est le guide de référence du langage de 4D. Il vous apprend à personnaliser votre base en utilisant les commandes et fonctions de 4D.
- Le manuel "Mode Développement" détaille les éditeurs et les outils disponibles dans l'environnement de développement.
- Le manuel "Autoformation" vous propose de suivre au pas à pas des exemples de création et d'utilisation de bases de données. Ces exercices vous permettent de vous familiariser rapidement avec les fonctionnalités et les concepts de 4D et 4D Server.
- Le "Manuel de référence 4D Server" est consacré à l'installation et la gestion de bases de données multi-utilisateurs avec 4D Server..

A propos de ce manuel

Ce manuel décrit le langage de 4D. Nous supposons que vous êtes familiarisé avec le vocabulaire élémentaire utilisé dans 4D tels que tables, champs ou formulaires. Avant de lire ce manuel, nous vous conseillons de :

- découvrir 4D à travers les exemples du manuel "Autoformation",
- commencer à créer vos propres bases de données, en vous référant au manuel "Mode Développement",
- élargir vos connaissances en étudiant les nombreuses bases de démonstration et d'exemple disponibles notamment sur le site Web de 4D (*http://www.4d.com*).

Conventions d'écriture

Dans ce manuel, diverses conventions d'écriture sont employées :

- à l'instar de l'éditeur de méthodes de 4D, les commandes sont écrites en majuscules et en caractères spéciaux : **FERMER DOCUMENT**. Les fonctions (commandes renvoyant une valeur) ont une lettre initiale en majuscule et sont écrites en minuscule : **Remplacer caracteres**.
- Dans la syntaxe des commandes, les caractères { } (accolades) indiquent des paramètres facultatifs. Par exemple, **SIECLE PAR DEFAUT (siècle{; anPivot})** signifie que le paramètre *anPivot* peut être omis lors de l'appel de la commande.
- Dans la syntaxe des commandes, le caractère | indique une alternative. Par exemple, **Table (numTable | unPtr)** indique que cette fonction accepte soit un numéro de table soit un pointeur comme paramètre.
- Dans certains exemples de cette documentation, une ligne de code peut se prolonger sur une plusieurs lignes, par manque de place. Toutefois, tapez ces exemples sans appuyer sur la touche Retour chariot, en une seule ligne de code.

Pour en savoir plus...

Si vous lisez ce manuel pour la première fois, reportez-vous à la section **Introduction**.

Introduction

Cette section présente le langage de 4D. Elle apporte des réponses aux questions suivantes :

- Qu'est-ce que le langage de 4D, et que peut-il vous apporter ?
- Comment utiliser les méthodes ?
- Comment développer une application avec 4D ?

Ces sujets sont évoqués d'un point de vue général ; chacun d'entre eux est traité plus en détail dans les autres sections de ce manuel.

Qu'est-ce qu'un langage ?

Le langage de 4D n'est pas très différent de celui que nous utilisons tous les jours. C'est une forme de communication servant à exprimer des idées, informer ou donner des instructions. Tout comme un langage parlé, celui de 4D se compose d'un vocabulaire, d'une grammaire et d'une syntaxe, que vous employez pour indiquer au programme comment gérer votre base et vos données..

Il n'est pas nécessaire de connaître entièrement le langage. Pour pouvoir vous exprimer en anglais, vous n'êtes pas obligé de connaître la totalité de la langue anglaise ; en réalité, un peu de vocabulaire suffit. Il en va de même pour 4D — il vous suffit de connaître un minimum du langage pour être productif, vous en apprendrez de plus en plus au fur et à mesure de vos besoins.

Pourquoi utiliser un langage ?

A première vue, un langage de programmation dans 4D peut sembler inutile. En effet, 4D propose en mode Développement des outils puissants et entièrement paramétrables, permettant d'effectuer une grande variété de tâches de gestion des données. Les opérations fondamentales telles que la saisie de données, les recherches, les tris et la création d'états sont effectuées facilement. De nombreuses autres possibilités sont aussi disponibles, telles que les filtres de validation des données, les aides à la saisie, la représentation graphique et la génération d'étiquettes.

Alors, pourquoi avons-nous besoin d'un langage ? Voici quelques-uns de ses principaux rôles :

- Automatiser les tâches répétitives : par exemple la modification de données, la génération d'états complexes et la réalisation de longues séries d'opérations ne nécessitant pas l'intervention d'un utilisateur.
- Contrôler l'interface utilisateur : vous pouvez gérer les fenêtres, les menus, contrôler les formulaires et les objets d'interface.
- Gérer les données de manière très fine : cette gestion inclut le traitement de transactions, les systèmes de validation complexes, la gestion multi-utilisateurs, l'utilisation des ensembles et des sélections temporaires.
- Contrôler l'ordinateur : vous pouvez contrôler les communications par le port série, la gestion des documents et des erreurs.
- Créer des applications : vous pouvez créer des bases de données personnalisées, faciles à utiliser, en mode Application.
- Ajouter des fonctionnalités au serveur Web 4D intégré : par exemple, vous pouvez créer des pages HTML dynamiques et les insérer parmi celles qui sont automatiquement créées par 4D lorsque les formulaires sont convertis.

Le langage vous permet de contrôler totalement la structure et le fonctionnement de votre base de données. 4D propose de puissants éditeurs "génériques", mais le langage vous offre la possibilité de personnaliser autant que vous voulez votre base de données.

Prendre le contrôle de vos données

Le langage de 4D vous permet de prendre le contrôle total de vos données, d'une manière à la fois puissante et élégante. Il reste d'un abord facile pour un débutant, et est suffisamment riche pour un développeur d'applications. Il permet de passer par étapes d'une simple exploitation des fonctions intégrées de 4D à une base entièrement personnalisée.

Les commandes du langage de 4D vous laissent la possibilité d'accéder aux éditeurs standard de gestion des enregistrements. Par exemple, lorsque vous utilisez la commande **CHERCHER**, vous accédez à l'éditeur de recherches (accessible en mode Développement via la commande **Chercher...** dans le menu **Enregistrements**. Vous pouvez, si vous le voulez, indiquer explicitement à la commande **CHERCHER** ce qu'elle doit rechercher. Par exemple, **CHERCHER([Personnes]Nom="Dupont")** trouvera toutes les personnes nommées Dupont dans votre base.

Le langage de 4D est très puissant — une commande remplace souvent des centaines, voire des milliers de lignes de code écrites dans les langages informatiques traditionnels. Et cette puissance s'accompagne d'une réelle simplicité: les commandes sont écrites en français. Par exemple, vous utilisez la commande **CHERCHER** pour effectuer une recherche ; pour ajouter un nouvel enregistrement, vous appelez la commande **AJOUTER ENREGISTREMENT**.

Le langage est organisé de telle manière que vous puissiez facilement accomplir n'importe quelle tâche. L'ajout d'un enregistrement, le tri d'enregistrements, la recherche de valeurs et les autres opérations de même type sont définies par des commandes simples et directes. Mais les routines peuvent également contrôler les ports série, lire des documents sur le disque, traiter des systèmes de transactions complexes, et plus encore.

Même les opérations les plus compliquées se définissent de manière relativement simple. Il serait inimaginable de les réaliser sans le langage de 4D. Cependant, même à l'aide de la puissance des commandes du langage, certaines tâches peuvent se révéler complexes et difficiles. Ce n'est pas l'outil lui-même qui fait le travail ; une tâche peut représenter en soi une difficulté, l'outil ne fait que faciliter son traitement. Par exemple, un logiciel de traitement de texte permet d'écrire un livre plus rapidement et plus facilement, mais il ne l'écrira pas à votre place. Le langage de 4D vous permet de gérer plus facilement vos données et d'appréhender les tâches ardues en toute confiance.

Est-ce un langage informatique "traditionnel" ?

Si vous êtes familier avec les langages informatiques traditionnels, ce paragraphe peut vous intéresser. Si ce n'est pas le cas, vous pouvez passer directement au paragraphe suivant.

Le langage de 4D n'est pas un langage informatique traditionnel. C'est un des langages les plus souples et les plus innovants que l'on puisse trouver aujourd'hui sur micro-ordinateur. Le langage a été conçu pour s'adapter à vous, et non l'inverse.

Pour utiliser les langages traditionnels, vous devez avant tout réaliser des maquettes détaillées. C'est même l'une des principales phases d'un développement. 4D vous permet de commencer à utiliser le langage à tout moment et dans n'importe quelle partie de votre base. Vous pouvez commencer par ajouter une méthode objet dans un formulaire, puis plus tard une ou deux méthodes formulaires. Comme votre base devient plus sophistiquée, vous pouvez ajouter une méthode projet contrôlée par un menu. Vous êtes totalement libre d'utiliser une petite partie du langage ou une partie plus étendue. Ce

n’est pas “tout ou rien”, comme c’est le cas dans beaucoup d’autres bases de données.

Les langages traditionnels vous obligent à définir et pré-déclarer vos objets sous une forme syntaxique rigide. Dans 4D, vous créez simplement un objet, comme un bouton, et vous l’utilisez. 4D gère automatiquement l’objet pour vous. Par exemple, pour utiliser un bouton, il suffit de le dessiner dans un formulaire et de lui donner un nom. Lorsque l’utilisateur clique sur le bouton, le langage déclenche automatiquement vos méthodes.

Les langages traditionnels sont souvent rigides et inflexibles, et exigent que les commandes soient saisies dans un style très formel et contraignant. Le langage de 4D rompt avec la tradition, au grand bénéfice de l’utilisateur.

Les méthodes, portes d'accès au langage

Une méthode est une série d’instructions qui provoquent l’accomplissement d’une tâche par 4D. Toute méthode contient une ou plusieurs lignes d’instructions. Chaque ligne d’instruction est composée d’éléments du langage.

Puisque vous avez lu le manuel "Autoformation" de 4D (vous l’avez lu, n’est-ce pas ?), vous avez déjà écrit et utilisé des méthodes objet et des méthodes projet.

Dans 4D, vous pouvez créer cinq types de méthodes : des méthodes objet, des méthodes formulaire, des méthodes table ou "triggers", des méthodes projet et des méthodes base.

- **Méthode objet :** méthode généralement courte utilisée pour contrôler un objet de formulaire.
- **Méthode formulaire :** méthode qui gère l’affichage ou l'impression d’un formulaire.
- **Méthode table/trigger :** méthode utilisée pour appliquer les règles de fonctionnement de votre base.
- **Méthode projet :** méthode s’appliquant à la totalité de la base. Les méthodes projet peuvent être associées à des commandes de menus, par exemple.
- **Méthode base :** méthode utilisée pour initialiser des valeurs ou effectuer des actions particulières à l'ouverture ou à la fermeture d'une base, ou lorsqu'un navigateur Web se connecte à une base publiée comme serveur Web sur un réseau Intranet ou sur Internet.

Les paragraphes suivants présentent chaque type de méthode et expliquent brièvement la manière dont vous pouvez les utiliser pour automatiser votre base.

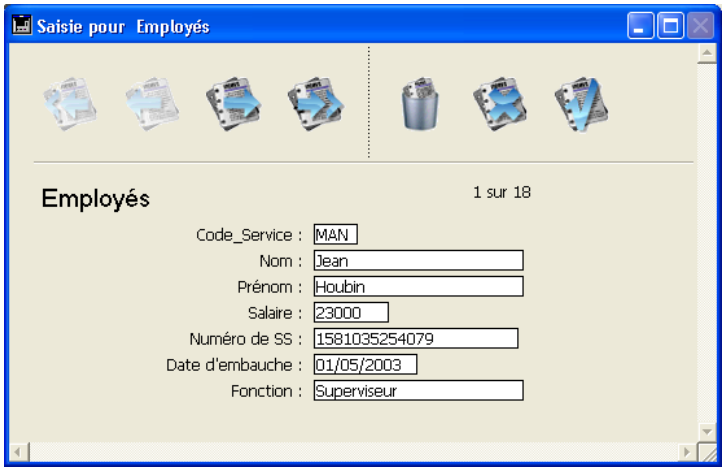
Démarrer avec les méthodes objet

Tout objet d’un formulaire pouvant déclencher une action — c’est-à-dire tout objet actif — peut être associé à une méthode objet. Une méthode objet surveille et gère l’objet actif lors de la saisie et de l’impression des données. Une méthode objet reste liée à un objet actif même lorsque l’objet est copié et collé. Ce principe vous permet de créer des librairies d’objets actifs réutilisables. La méthode objet s’exécute lorsque c’est nécessaire.

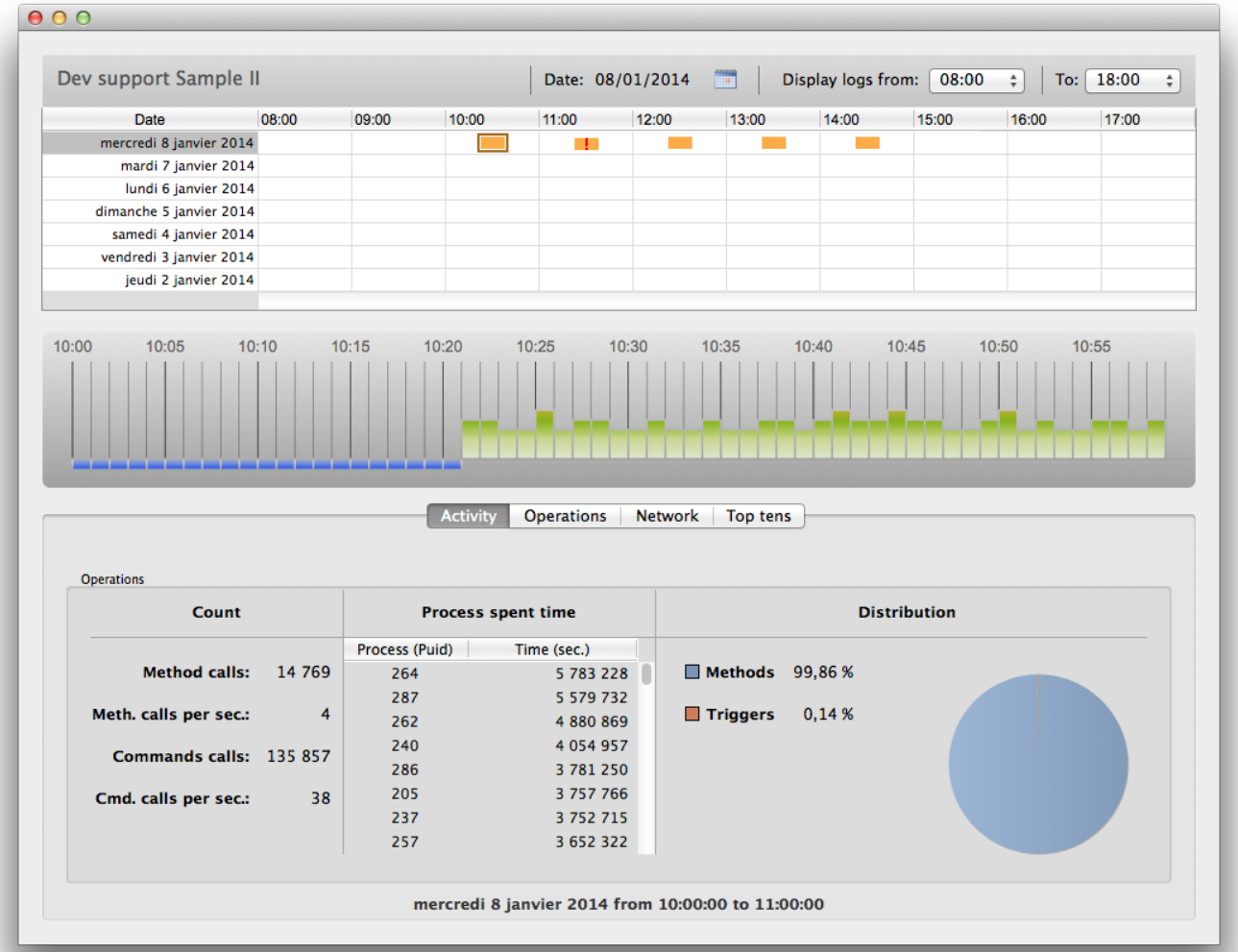
Les méthodes objet sont les outils de base pour la gestion de l’interface utilisateur. L’interface utilisateur est l’ensemble des moyens et des conventions par lesquels l’ordinateur communique avec l’utilisateur. L’interface utilisateur est la porte par laquelle vous entrez dans votre base de données. L’objectif est de la rendre aussi simple d'emploi que possible. L’interface utilisateur doit faire en sorte que l’interaction avec l’ordinateur soit agréable, que l’utilisateur l’apprécie ou, mieux, ne la remarque même pas.

Il y a deux types principaux d’objets actifs dans un formulaire : les objets actifs de saisie, d’affichage et de stockage des données, tels que les champs et les sous-champs, et les objets actifs de contrôle comme les zones de saisie, les boutons, les listes déroulantes et les thermomètres.

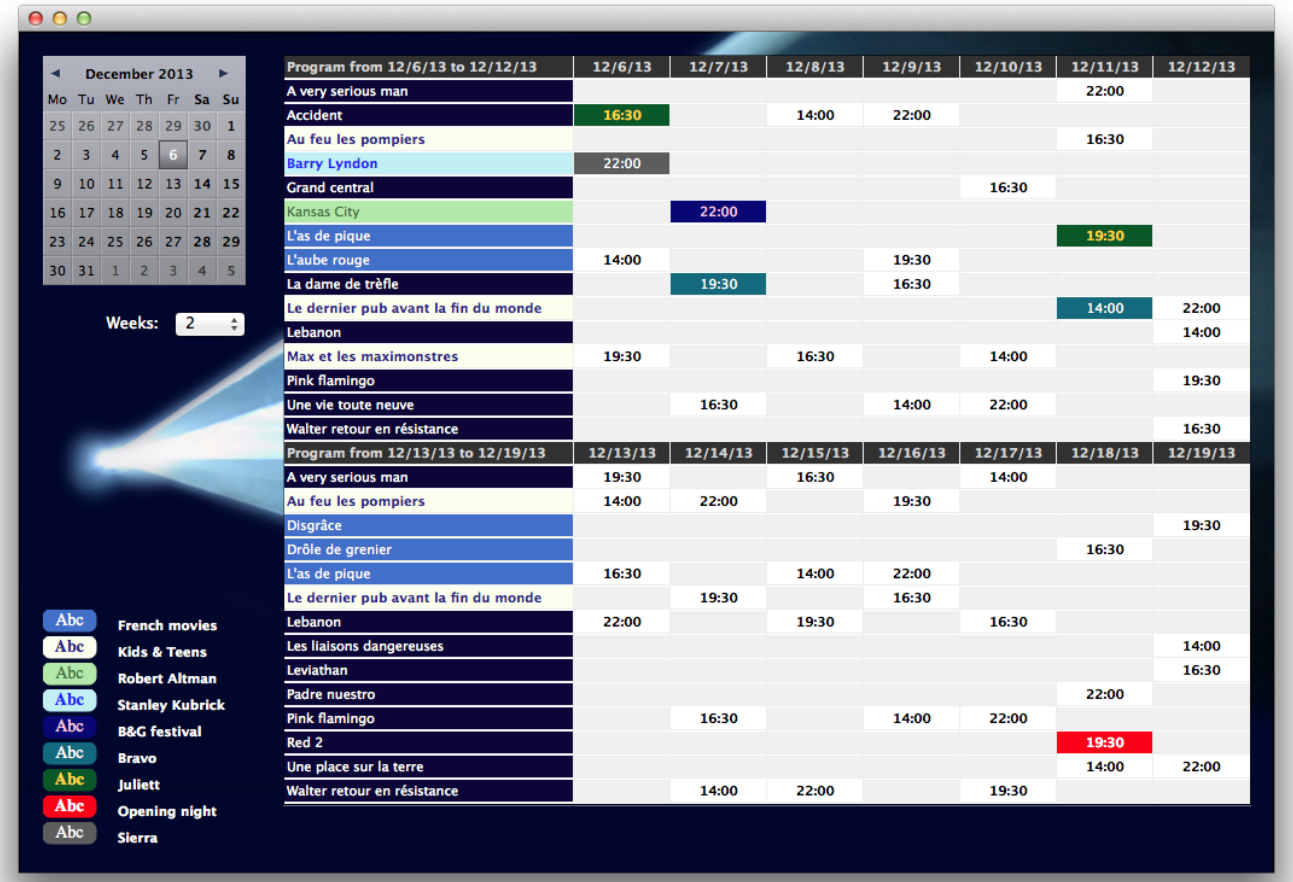
4D vous permet de construire des formulaires sobres et classiques, tel que celui présenté ci-dessous :



4D vous permet également de construire des formulaires comportant de multiples éléments de contrôle, comme celui-ci :



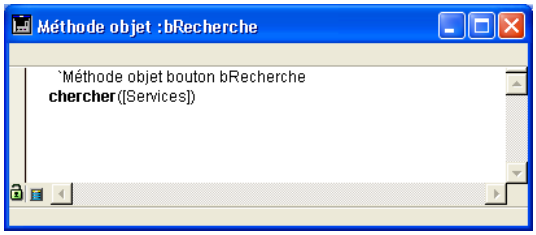
Vous pouvez aussi créer des formulaires originaux en laissant libre cours à votre imagination et en utilisant des éléments graphiques exubérants. Vous n'êtes limité que par votre imagination ou par les souhaits des commanditaires de l'application :



Lorsque vous construisez des formulaires, rappelez-vous que tous les objets actifs disposent de systèmes de contrôle intégrés, comme les intervalles de valeurs et les filtres pour les zones de saisie, ou les actions automatiques pour les objets, menus et boutons. Essayez toujours d'utiliser ces systèmes intégrés avant de recourir aux méthodes objet. Ils sont comparables aux méthodes objet dans la mesure où ils restent associés aux objets actifs et ne sont exécutés que lorsque l'objet est sollicité. En général, vous utiliserez une combinaison d'aides intégrées et de méthodes objet pour contrôler l'interface utilisateur.

Typiquement, une méthode associée à un objet actif de saisie aura un rôle de gestion des données spécifiquement lié au champ ou à la variable. La méthode peut effectuer un contrôle ou un formatage des valeurs, ou des calculs. Elle peut également récupérer des informations en provenance d'autres tables. Bien entendu, certaines de ces tâches peuvent aussi être exécutées par l'intermédiaire des systèmes intégrés de contrôle de saisie des objets. N'utilisez les méthodes objet que lorsque l'opération à effectuer est plus complexe que ce que permettent ces systèmes. Pour plus d'informations sur les systèmes intégrés de contrôle de saisie, reportez-vous au manuel "Mode Développement" de 4D.

Des méthodes objet peuvent aussi être associées aux objets actifs de contrôle tels que les boutons. Ces objets sont essentiels pour la navigation au sein de votre base. Les boutons vous permettent de vous déplacer d'un enregistrement à l'autre, de changer de formulaire, d'ajouter et d'effacer des données. Ces objets actifs simplifient l'utilisation d'une base et réduisent le temps d'apprentissage. Les boutons disposent également de systèmes d'aides intégrés et, comme pour la saisie de données, il est préférable de les utiliser avant d'écrire des méthodes objet. Les méthodes vous permettent d'associer à vos objets des actions qui n'existent pas en standard. Dans l'exemple ci-dessous, la méthode objet du bouton affichera l'éditeur de recherches lorsque l'utilisateur cliquera dessus.



A mesure que vous vous familiariserez avec les méthodes objet, vous pourrez créer des librairies d'objets avec leurs méthodes associées. Vous pourrez copier et coller ces objets entre différents formulaires, tables et bases.

Contrôler des formulaires à l'aide des méthodes formulaire

Tout comme une méthode objet est associée à un objet actif d'un formulaire, une méthode formulaire est associée à un formulaire. Chaque formulaire peut comporter au plus une méthode formulaire.

Un formulaire est un modèle dans lequel vous pouvez visualiser vos données. Les formulaires vous permettent de présenter les données à l'utilisateur de différentes manières. Grâce aux formulaires, vous pouvez créer des écrans de saisie et des états récapitulatifs attractifs et faciles à utiliser. Une méthode formulaire contrôle et gère l'utilisation d'un formulaire spécifique, à la fois pour la saisie et l'impression des données.

Une méthode formulaire gère un formulaire à un plus haut niveau que les méthodes objet. Une méthode objet n'est exécutée que lorsque l'objet est utilisé, alors qu'une méthode formulaire est exécutée lorsque n'importe quel objet du formulaire est activé. Les méthodes formulaire sont généralement utilisées pour contrôler l'interaction entre les différents objets et le formulaire dans son ensemble.

Comme les formulaires peuvent être utilisés de nombreuses manières, il est utile de contrôler ce qui se passe lorsque votre formulaire est en cours d'utilisation. Pour cela, 4D met à votre disposition un grand nombre **d'événements formulaires**. Ces événements vous indiquent précisément ce qui se produit dans le formulaire. Chaque type d'événement (par exemple un clic, un double-clic, une touche du clavier enfoncée...) active ou désactive l'exécution de la méthode formulaire ainsi que les méthodes des objets du formulaire.

Pour plus d'informations sur les formulaires, les objets, les événements et les méthodes, reportez-vous à la description de la commande **Evenement formulaire**.

Réguler le fonctionnement de votre base à l'aide des méthodes table/triggers

Un trigger est une méthode associée à une table, c'est la raison pour laquelle on peut également parler de méthode table. Les triggers sont automatiquement appelés par le moteur de base de données de 4D à chaque fois qu'une action (ajout, suppression, modification et chargement) est effectuée sur les enregistrements d'une table. Les triggers peuvent empêcher que des opérations interdites soient exécutées avec les enregistrements de votre base. Par exemple, dans un système de facturation, vous ne voulez pas qu'un utilisateur puisse créer des factures sans avoir spécifié le client à qui elle est destinée. Les triggers sont des outils puissants de contrôle des opérations possibles avec les tables, ainsi que de prévention des risques de pertes accidentelles de données. Vous pouvez écrire des triggers très simples, puis les rendre de plus en plus sophistiqués.

Pour plus d'informations sur les triggers, reportez-vous à la section **Présentation des triggers**.

Utiliser des méthodes projet dans toute la base

A la différence des triggers, des méthodes formulaire et des méthodes objet, qui sont associées à des tables, des formulaires ou des objets actifs particuliers, les méthodes projet sont accessibles depuis n'importe quelle partie de votre base. Les méthodes projet sont réutilisables et peuvent être appelées par d'autres méthodes. Vous pouvez ainsi effectuer plusieurs fois une opération similaire sans devoir écrire plusieurs méthodes.
Vous pouvez appeler des méthodes projet depuis n'importe quelle partie de la base — autres méthodes projet, méthodes objet, méthodes formulaire... Lorsque vous appelez une méthode projet, elle s'exécute comme si elle avait été écrite à l'endroit d'où elle a été appelée. Les méthodes projet appelées depuis d'autres procédures sont appelées sous-routines.
Il y a un autre moyen d'utiliser les méthodes projet : les associer à des commandes de menu. Lorsque vous associez une méthode projet à une commande de menu, la méthode est exécutée lorsque la commande est sélectionnée par l'utilisateur. Vous pouvez considérer la commande de menu comme un appel à une méthode projet.

Gérer les sessions de travail avec les méthodes base

De la même manière que les méthodes objet et formulaire sont exécutées lorsque des événements se produisent dans un formulaire, il existe des méthodes associées à la base qui sont exécutées lorsqu'un événement de type "session de travail" se produit : ce sont les **méthodes base**. Par exemple, à chaque fois que vous ouvrez une base de données, vous voulez initialiser certaines variables qui seront utilisées pendant toute la session de travail ; pour cela, vous pouvez initialiser vos variables dans la **Méthode base Sur ouverture**, qui est automatiquement exécutée par 4D lorsque vous ouvrez la base.

Pour plus d'informations sur les méthode base, reportez-vous à la section **Méthodes base**.

Développer votre base de données

La phase de développement est celle pendant laquelle vous personnalisez votre base à l'aide du langage et des autres fonctions intégrées.

En créant simplement une base, vous avez déjà franchi les premières étapes de l'utilisation du langage. Chaque élément d'une base — les tables et les champs, les formulaires et leurs objets — est automatiquement relié au langage. Le langage de 4D les "reconnaît" tous.

Votre première utilisation du langage pourrait être de créer une méthode objet ou formulaire pour contrôler la saisie de données. Par la suite, vous pourrez définir une méthode formulaire pour gérer l'affichage. Lorsque la base deviendra plus complexe, vous ajouterez une barre de menus avec des méthodes projet pour la personnaliser entièrement.

Comme tous les autres aspects de 4D, le développement est une phase très souple. Il n'y a aucun cheminement précis à suivre — vous pouvez développer de la manière qui vous semble la plus pratique. Bien entendu, cette phase comprend des étapes générales.

- L'implémentation : définition de la structure et des fonctionnalités de la base en mode Développement.
- Le test : réalisation d'essais de la base et de test de chaque élément personnalisé en mode Test application.
- L'utilisation : lorsque la base est entièrement personnalisée, exécution en mode Application.
- Les corrections : si des erreurs sont détectées, retour au mode Développement pour les corriger.

Des outils particuliers d'aide au développement, n'apparaissant que lorsque c'est nécessaire, sont intégrés à 4D. A mesure que vous utiliserez le langage de manière de plus en plus intensive, vous vous apercevrez que ces outils facilitent grandement le développement. Par exemple, l'éditeur de méthodes traite automatiquement les éventuelles erreurs de saisie et formate votre travail à la volée ; l'interpréteur (le moteur qui exécute le langage) intercepte les erreurs de syntaxe et vous les désigne ; enfin, le débogueur vous permet de contrôler très précisément l'exécution de vos méthodes afin de détecter toute erreur de développement.

Construire des applications

Vous connaissez maintenant les opérations que l'on peut réaliser avec une base de données — saisie, recherches, tris et créations d'états... Vous avez pu effectuer ces actions depuis le mode Développement, à l'aide des menus et des éditeurs standard.

Lorsque vous utilisez une base de données, vous répétez certaines séquences de tâches. Par exemple, dans une base de contacts personnels, vous pouvez rechercher des contacts, les trier par nom, puis imprimer un état à chaque fois que des informations ont été modifiées. Ces opérations ne sont pas difficiles à réaliser, mais elles peuvent prendre beaucoup de temps lorsqu'il faut les faire 20 fois. De plus, si vous n'avez pas utilisé la base pendant deux semaines, il se peut que vous ayez oublié certaines étapes nécessaires à la création d'un état.
Dans les méthodes, les étapes s'enchaînent les unes après les autres. Ainsi, une seule commande effectue automatiquement toutes les tâches qui lui sont liées. Par conséquent, vous n'avez pas à vous préoccuper des opérations particulières à réaliser.

Dans vos applications, les menus créés permettent d'effectuer des tâches répondant précisément aux besoins de la personne qui utilise la base. Une application se compose de tous les éléments de votre base : la structure, les formulaires, les différentes méthodes, les menus et les mots de passe.

Vous pouvez compiler vos bases et créer des applications Windows et Mac OS autonomes. La compilation des bases de données accélère l'exécution du langage, protège les bases, et vous permet de créer des applications totalement indépendantes. Le compilateur intégré vérifie également la syntaxe ainsi que les types des variables dans les méthodes, et contrôle donc la cohérence de base.

Les applications que vous créez peuvent aller du plus simple (un menu unique permettant de saisir des noms et d'imprimer un état) au plus complexe (un système de facturation ou de gestion des stocks). Les types d'utilisation des applications sont illimités. Généralement, une application grandit progressivement depuis une base de données en mode Développement jusqu'à un logiciel entièrement contrôlé par des menus et des formulaires personnalisés.

Pour en savoir plus...

- Le développement d'applications peut être aussi simple ou aussi complexe que vous le voulez. Pour plus d'informations sur le processus de construction d'une application, reportez-vous à la section **Construire une application 4D**.
- Si vous débutez avec 4D, reportez-vous aux sections **Présentation du langage** pour découvrir les principes de fonctionnement du langage 4D.

Commencez par la section **Introduction au langage 4D**.

Une application 4D est une base de données conçue pour répondre spécifiquement à des besoins précis. Une application comporte une interface utilisateur destinée à faciliter son utilisation. Toutes les fonctions qu'elle propose sont directement liées (et se limitent) à son champ d'action. 4D vous permet de créer des applications de manière plus confortable et plus aisée que si vous utilisiez des langages traditionnels.
Parmi la grande variété d'applications que 4D peut créer, citons :

- un système de facturation,
- un système de gestion des stocks,
- un système de comptabilité,
- un système de paie,
- un gestion des ressources humaines,
- un système de traitement des prospects,
- une base de données accessible par Internet ou Intranet.

Il est parfaitement envisageable qu'une seule application 4D gère tous ces systèmes. Ce type d'applications correspond à une utilisation plutôt traditionnelle des bases de données. De surcroît, les outils proposés par 4D vous permettent de créer des applications originales, telles que, par exemple :

- un système de gestion documentaire,
- un système graphique de gestion d'images,
- une application de publication de catalogues,
- un système de contrôle et de commande d'un dispositif externe,
- un système de messagerie électronique (E-mail),
- un système multi-utilisateurs de gestion de planning,
- un catalogue recensant les éléments d'une collection de vidéos ou de disques.

Typiquement, une application peut commencer par être une simple base de données exploitée en mode Développement. Cette base "se transforme" en application à mesure qu'elle est personnalisée. La caractéristique majeure d'une application est la dissimulation à l'utilisateur des systèmes internes de gestion des fonctions de la base. La gestion de la base est automatisée, et l'utilisateur se sert de menus personnalisés pour exécuter des tâches spécifiques.

Lorsqu'une base 4D est exploitée en mode Développement, vous devez connaître les étapes à atteindre pour obtenir le résultat recherché. Dans une application 4D, c'est le mode Application qui est utilisé. Dans ce mode, vous devez gérer vous-même toutes les fonctions qui sont automatiques dans le mode Développement, c'est-à-dire :

- Navigation parmi les tables : la Liste des tables, la commande Dernière tables utilisées ou les boutons de navigation ne sont pas accessibles à l'utilisateur. Vous pouvez utiliser les commandes de menus et les méthodes pour contrôler la navigation parmi les tables.
- Menus : en mode Application, seuls les menus Fichier (comportant la commande Quitter), Edition, Mode et Aide (ainsi que le menu application sous Mac OS) sont affichés par défaut. Si l'application nécessite d'autres menus, vous devez les créer vous-même et les gérer à l'aide de méthodes 4D ou d'actions standard.
- Editeurs : les éditeurs, tels que les éditeurs de recherches et de tris, ne sont plus automatiquement accessibles en mode Application. Si vous voulez qu'ils restent disponibles, vous devez les appeler en utilisant les commandes du langage.

Les paragraphes suivants fournissent des exemples d'automatisation de l'utilisation d'une base à l'aide du langage.

Mode Application : un exemple

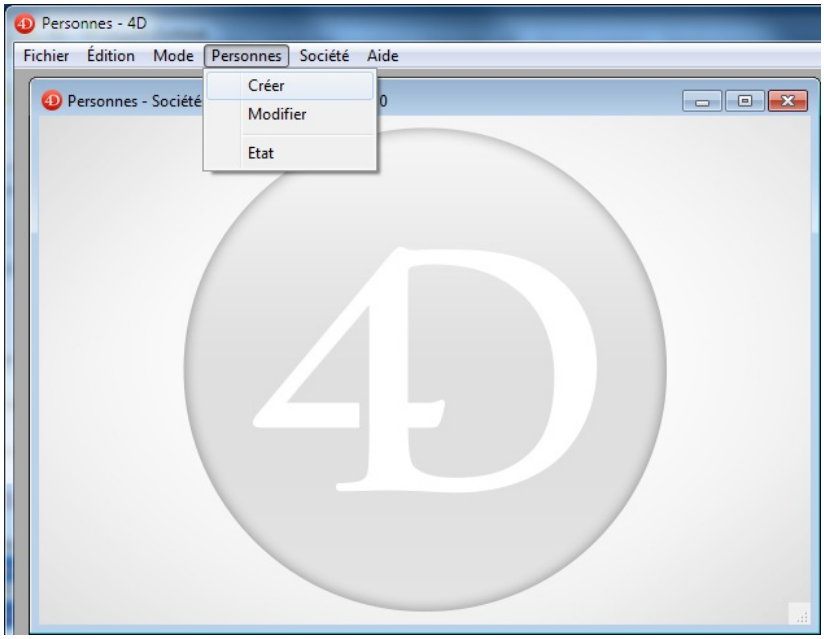
Les menus personnalisés sont les éléments d'interface élémentaires d'une application. La création de menus est très simple — il suffit d'associer une méthode ou une action standard à chaque commande de menu dans l'éditeur de menus.

Les menus personnalisés facilitent l'apprentissage et l'utilisation d'une base de données.

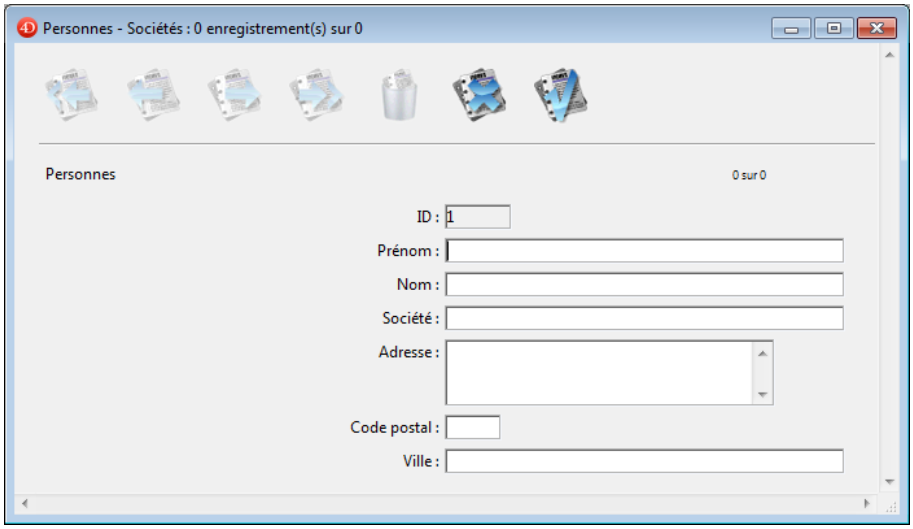
Le paragraphe ci-dessous décrit le point de vue de l'utilisateur lorsqu'il choisit une commande de menu. Le paragraphe suivant détaille ensuite ce qui se produit réellement au cœur l'application ainsi que le travail de conception qui a été effectué. Bien que l'exemple soit simple, il apparaît clairement que la base est plus facile à apprendre et à utiliser. L'utilisateur n'a accès qu'aux éléments correspondant à ses besoins plutôt qu'aux outils "génériques" et aux commandes de menu du mode Développement.

Le point de vue de l'utilisateur

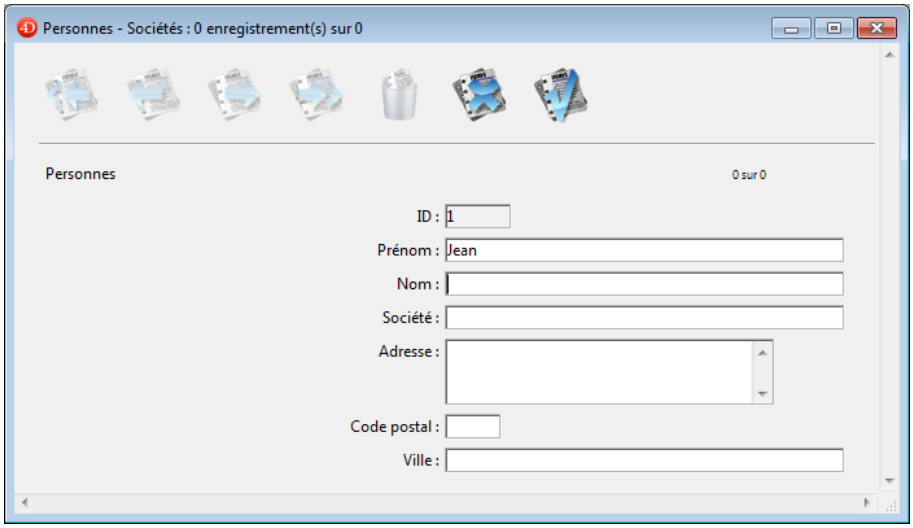
L'utilisateur sélectionne une commande de menu personnalisé appelée **Créer** pour ajouter une nouvelle personne dans la base de données.



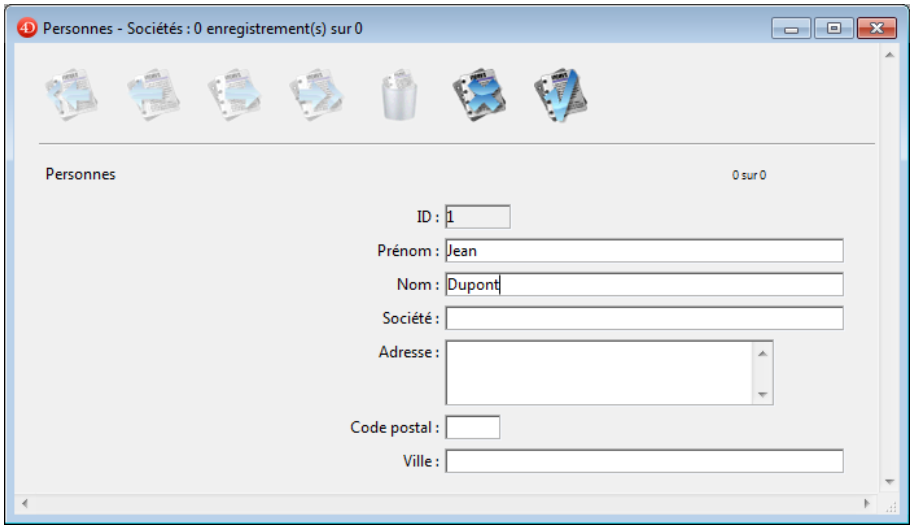
Le formulaire entrée de la table [Personnes] s'affiche.



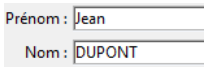
L'utilisateur saisit le prénom de la personne et appuie sur la touche **Tabulation** pour passer au champ suivant.



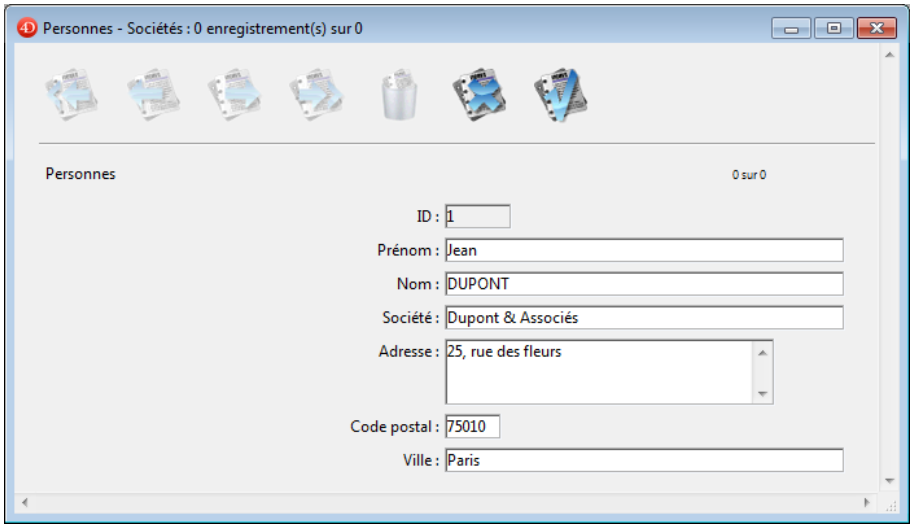
L'utilisateur saisit le nom de la personne.



L'utilisateur appuie sur la touche **Tabulation** pour passer au champ suivant : le nom est converti en lettres majuscules.



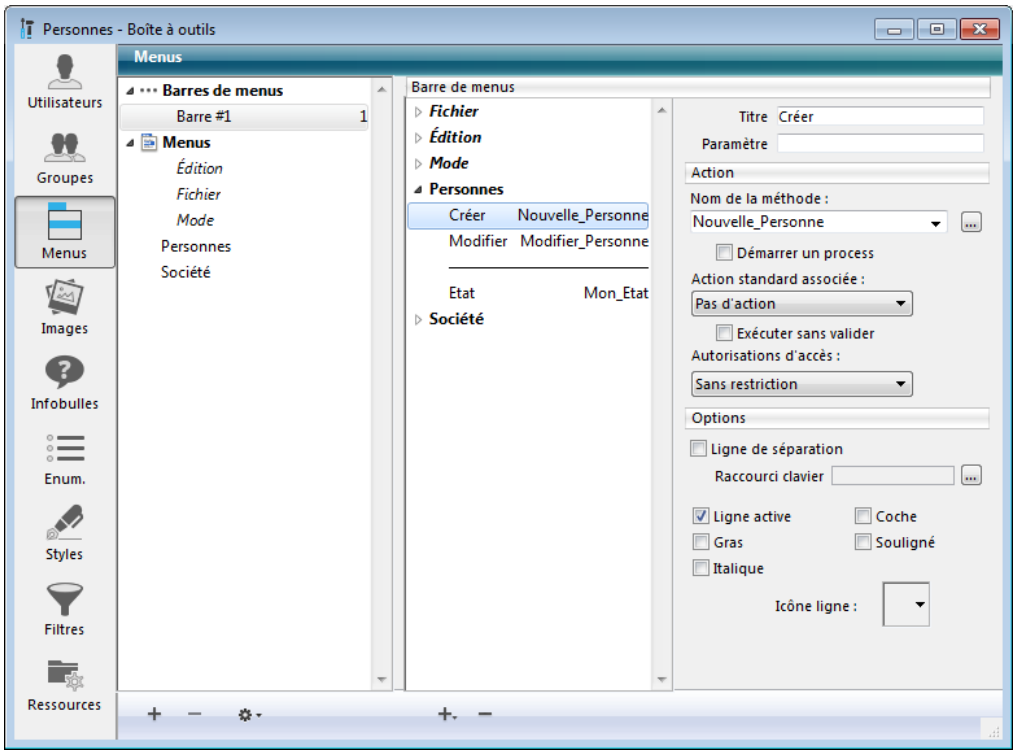
Une fois la saisie terminée, l'utilisateur clique sur le bouton de validation (généralement le dernier dans la barre de boutons).



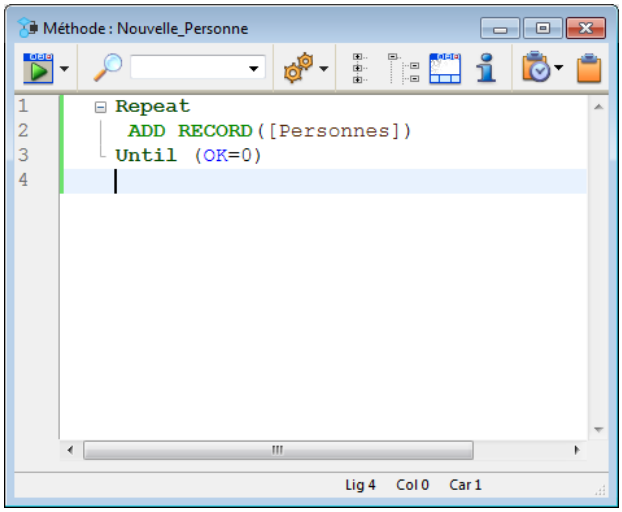
Un autre enregistrement vide s'affiche. L'utilisateur clique sur le bouton **Annuler** (celui qui comporte une croix) pour terminer la "boucle de saisie des données". L'utilisateur retourne à l'écran principal.

Les coulisses

La barre de menus a été créée en mode Développement, à l'aide de l'éditeur de menus.



La commande de menu **Créer** est associée à une méthode projet appelée ***Nouvelle Personne***. Cette méthode a été créée, en mode Développement, dans l'éditeur de méthodes. Lorsque l'utilisateur sélectionne cette commande de menu, la méthode ***Nouvelle Personne*** s'exécute.



La boucle **Repeter...Jusque** à l'intérieur de laquelle se trouve la commande **AJOUTER ENREGISTREMENT** provoque exactement les mêmes effets que la commande de menu **Nouvel enregistrement** en mode Développement. Elle affiche le formulaire entrée courant, de manière à ce que l'utilisateur puisse saisir un nouvel enregistrement. Lorsque l'utilisateur valide l'enregistrement, un autre enregistrement vide apparaît. Cette boucle **AJOUTER ENREGISTREMENT** continue de s'exécuter jusqu'à ce que l'utilisateur clique sur le bouton **Annuler**.

Lorsque l'utilisateur remplit un enregistrement, les actions suivantes sont déclenchées :

- Comme il n'y a pas de méthode objet associée au champ **Prénom**, rien ne s'exécute.
- Une méthode est associée au champ **Nom**. Cette méthode a été créée, en mode Développement, à l'aide des éditeurs de formulaires et de méthodes. Elle exécute l'instruction suivante :

```
[Personnes]Nom:=Majusc([Personnes]Nom)
```

Cette ligne convertit le champ "Nom" en caractères majuscules.

Une fois qu'un enregistrement a été saisi, lorsque l'utilisateur clique sur le bouton d'annulation dans le formulaire suivant, la variable système OK prend la valeur zéro, ce qui constitue la condition d'arrêt de l'exécution de la boucle **AJOUTER ENREGISTREMENT**.

Comme il n'y a pas d'autres instructions à exécuter, la méthode **Nouvelle Personne** stoppe son exécution et rend la main à la barre de menus principale.

Comparer une application 4D avec le mode Développement

Comparons la manière dont une même tâche est effectuée en mode Développement et à l'aide du langage.

Examinons une opération courante :

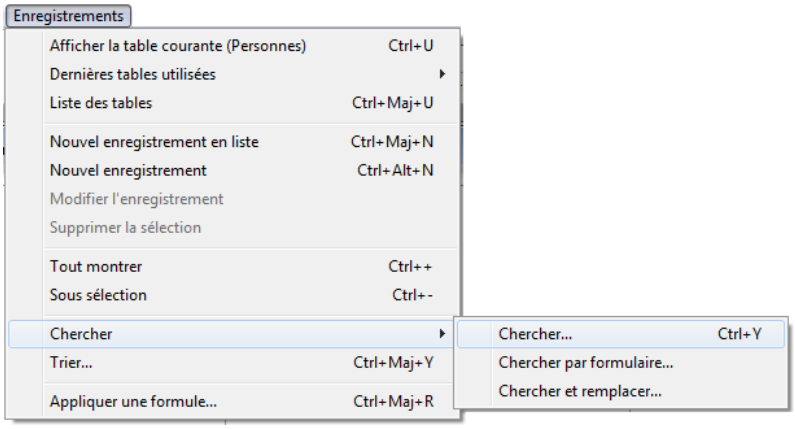
- Sélectionner un groupe d'enregistrements,
- Le trier,
- Imprimer un état.

Le paragraphe ci-dessous, "Exploiter une base en mode Développement", traite de la réalisation de ces tâches à partir du mode Développement. Le paragraphe suivant, "Exploiter une application 4D avec les éditeurs intégrés", traite de la réalisation des mêmes tâches à partir du mode Application.

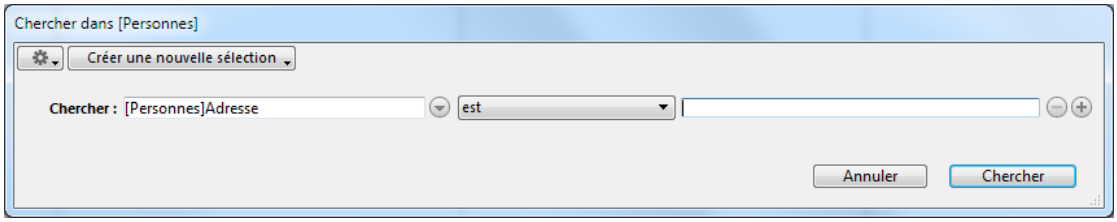
Notez que, bien que les mêmes opérations sont effectuées dans les deux cas, les étapes dans le second paragraphe sont automatisées par programmation.

Exploiter une base en mode Développement

L'utilisateur choisit la commande **Chercher>Chercher...** dans le menu **Enregistrements**.

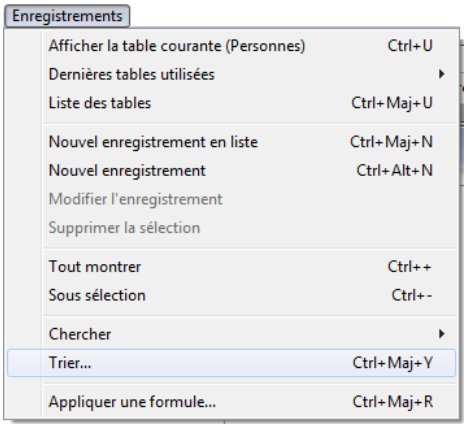


L'éditeur de recherches s'affiche :

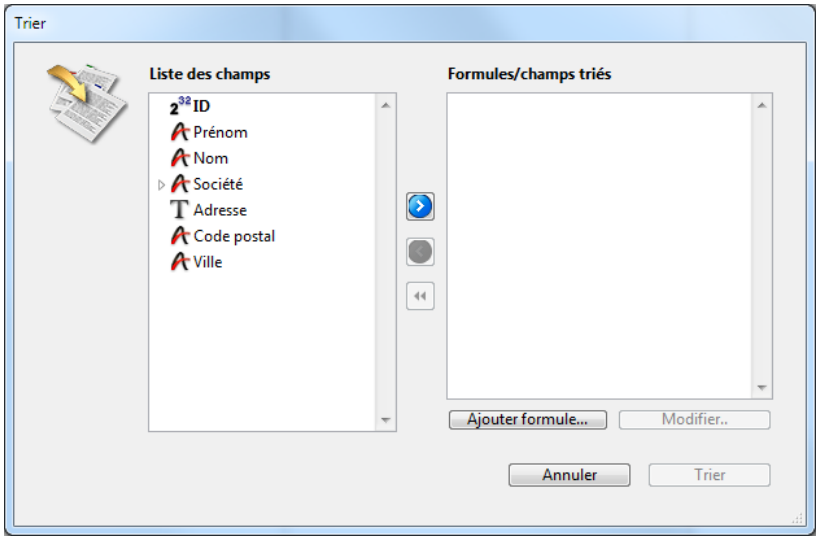


L'utilisateur définit ses critères de recherche et clique sur le bouton **Chercher**. La recherche s'effectue.

L'utilisateur choisit la commande **Trier...** dans le menu **Enregistrements**.



L'éditeur de tris s'affiche.



L'utilisateur définit ses critères de tri et clique sur le bouton **Trier**. Le tri est effectué.

Puis, pour imprimer les enregistrements, les étapes suivantes sont nécessaires :

- L'utilisateur choisit la commande **Imprimer** dans le menu **Fichier**.
- Les boîtes de dialogue de l'imprimante sélectionnée s'affichent. L'utilisateur fixe ses paramètres, puis l'état est imprimé.

Exploiter une application 4D avec les éditeurs intégrés

Examinons maintenant comment ces opérations peuvent être effectuées en mode Application.

L'utilisateur choisit la commande **Etat** dans le menu **Personnes**.

Même à ce stade, l'utilisation d'une application apparaît plus facile. L'utilisateur n'a pas besoin de savoir qu'il faut commencer par effectuer une recherche.

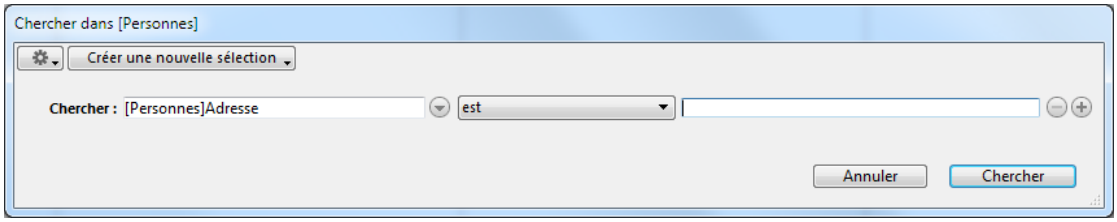
Une méthode appelée **Mon Etat** est associée à la commande de menu. Elle se présente ainsi :

```
CHERCHER([Personnes])
TRIER([Personnes])
FORM FIXER SORTIE([Personnes];"Etat")
IMPRIMER SELECTION([Personnes])
```

La première ligne est exécutée :

```
CHERCHER([Personnes])
```

L'éditeur de recherches s'affiche.



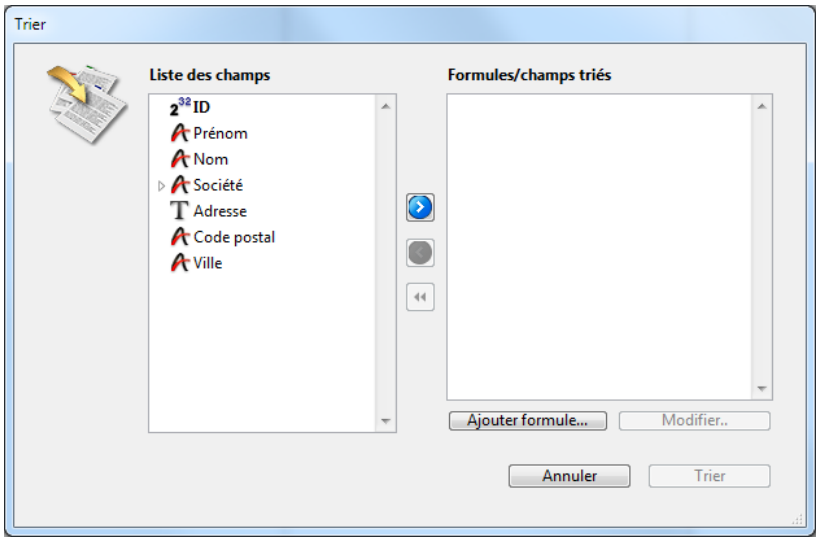
L'utilisateur définit ses critères de recherche et clique sur le bouton **Rechercher**. La recherche s'effectue.

La deuxième ligne de la méthode **Mon Etat** est exécutée :

```
TRIER([Personnes])
```

Vous notez que l'utilisateur n'avait pas besoin de savoir que le tri était la deuxième étape.

La boîte de dialogue de tri s'affiche.



L'utilisateur définit ses critères de tri et clique sur le bouton **Trier**. Le tri est effectué.

La troisième ligne de **Mon Etat** est exécutée :

```
FORM FIXER SORTIE([Personnes];"Etat")
```

Une fois de plus, il n'est pas nécessaire que l'utilisateur sache ce qu'il faut faire ensuite. Le cheminement est déjà tracé.

La dernière ligne de la méthode **Mon Etat** est exécutée :

```
IMPRIMER SELECTION([Personnes])
```

Les boîtes de dialogue de l'imprimante sélectionnée s'affichent. L'utilisateur fixe ses paramètres, puis l'état est imprimé.

Automatiser davantage l'application

Les mêmes commandes que celles qui ont été employées dans la comparaison précédente peuvent être utilisées pour automatiser encore plus la base de données.

Examinons la nouvelle version de la méthode **Mon Etat**.

L'utilisateur choisit **Etat** dans le menu **Personnes**. La méthode appelé **Mon Etat2** est associée à la commande de menu :

```
CHERCHER([Personnes];[Personnes]Entreprise="Dupont  &  Associés")
TRIER([Personnes];[Personnes]Nom;>;[Personnes]Prénom;>)
FORM FIXER SORTIE([Personnes];"Etat")
IMPRIMER SELECTION([Personnes];*)
```

La première ligne est exécutée :

```
CHERCHER([Personnes];[Personnes]Entreprise="Dupont  &  Associés")
```

L'éditeur de recherches ne s'affiche pas. Au lieu de cela, la recherche est définie et effectuée par la commande **CHERCHER**. L'utilisateur n'a rien à faire.

La deuxième ligne est exécutée :

```
TRIER([Personnes];[Personnes]Nom;>;[Personnes]Prénom;>)
```

La boîte de dialogue de tri n'est pas affichée, et le tri est immédiatement effectué. Une fois encore, aucune intervention de l'utilisateur n'est nécessaire.

Les dernières lignes de la méthode **Mon Etat2** sont exécutées :

```
FORM FIXER SORTIE([Personnes];"Etat")
IMPRIMER SELECTION([Personnes];*)
```

Les boîtes de dialogue standard d'impression ne sont pas affichées. La commande **IMPRIMER SELECTION** comporte en effet le paramètre optionnel

astérisque (*), ce qui lui indique d'utiliser les paramètres d'impression en vigueur au moment où le formulaire d'état a été créé. L'état est imprimé.

Les avantages de cette automatisation accrue sont les suivants :

- La recherche est effectuée automatiquement : cela évite que les utilisateurs sélectionnent de mauvais critères lorsqu'ils construisent leur recherche.
- Le tri est effectué automatiquement : cela évite que les utilisateurs sélectionnent de mauvais critères lorsqu'ils construisent leur tri.
- L'impression est effectuée automatiquement : cela évite que les utilisateurs sélectionnent un formulaire inadapté.
- Vous évitez à l'utilisateur d'avoir à définir des options dans les trois boîtes de dialogue.

Aides au développement d'applications 4D

A mesure que vous progresserez dans votre développement d'une application 4D, vous allez découvrir de nombreuses fonctionnalités que vous n'aviez pas vues lorsque vous avez commencé.
Mais vous pouvez aller encore plus loin, en ajoutant des outils et des plug-ins dans votre environnement 4D..

Plug-ins 4D

4D fournit plusieurs plug-ins permettant d'augmenter les capacités de vos applications, notamment :

- **4D Write** : Traitement de textes
- **4D View** : Tableur et éditeur de listes.
- **4D Internet Commands** (intégré) : Utilitaires de communication via Internet

- **4D ODBC Pro** : Connectivité bas niveau via ODBC
- **4D for OCI** : Connectivité avec ORACLE Call Interface

Pour plus d'informations, contactez 4D ou un partenaire 4D, ou visitez notre site Web :

http://www.4D.com/fr

La communauté 4D et les produits des partenaires 4D

Il existe une communauté internationale très active organisée autour de 4D, composée de groupes d'utilisateurs, de forums électroniques et de partenaires 4D, qui développe des **produits liés à 4D**.

Vous pouvez vous inscrire sur le forum public d'aide et d'échange de 4D à l'adresse suivante :

http://forums.4D.fr

La communauté 4D vous fournira de nombreux conseils et solutions, ainsi que des produits tiers vous permettant d'augmenter votre productivité. Vous aurez accès à une foule d'informations et d'astuces qui vous feront économiser votre temps et votre énergie.

Présentation du langage

Le langage de 4D est constitué de différents éléments, vous permettant d'effectuer de multiples opérations et de gérer vos données.

- **Types de données** : Catégories de données stockées dans une base. Ce point est traité dans le paragraphe suivant. Pour une description détaillée, référez-vous à la section **Types de données**.
- **Variables** : Adresses de stockage temporaires de données en mémoire. Pour une description détaillée, référez-vous à la section **Variables**.
- **Opérateurs** : Symboles effectuant un calcul entre deux valeurs. Ce point est traité dans les paragraphes suivants. Pour une description détaillée, référez-vous à la section **Opérateurs** et à ses sous-sections.
- **Expressions** : Combinaisons d'éléments du langage ayant pour résultat le renvoi d'une valeur. Ce point est traité dans les paragraphes suivants.
- **Commandes** : Instructions intégrées effectuant une opération. Toutes les commandes de 4D (par exemple **AJOUTER ENREGISTREMENT**) sont décrites dans ce manuel, groupées par thème. Lorsque c'est nécessaire, les thèmes comprennent une section d'introduction. Vous pouvez utiliser des Plug-ins 4D pour ajouter de nouvelles commandes à votre environnement de développement 4D. Par exemple, une fois que vous avez installé le plug-in 4D Write dans votre base de données, vous pouvez utiliser les commandes de 4D Write pour créer et manipuler des fichiers de traitement de texte.
- **Constantes prédéfinies** : Valeurs constantes accessibles par un nom. Par exemple, XML DATA est une constante (valeur 6). Les constantes prédéfinies permettent d'écrire un code plus lisible. Les constantes sont décrites avec les commandes qui les utilisent, et sont intégralement répertoriées dans la section **Liste des thèmes de constantes** .
- **Méthodes** : Instructions que vous écrivez à l'aide de tous les éléments du langage décrits ci-dessus. Pour une description détaillée, référez-vous à la section **Méthodes** et à ses sous-sections.

Cette section présente les **Types de données**, les **Opérateurs** et les **Expressions**. Pour une description d'un autre élément, reportez-vous aux sections précédemment citées.

De plus :

- Les éléments du langage tels que les variables sont identifiés par leur nom. Pour une description détaillée des règles de définition des noms d'objets, reportez-vous à la section **Nommer les objets du langage 4D**.
- Pour plus d'informations sur les variables tableaux, reportez-vous à la section **Tableaux**.
- Pour plus d'informations sur les variables BLOB, reportez-vous à la section **Commandes du thème BLOB**.
- Si vous avez l'intention de compiler votre base, reportez-vous à la section **Commandes du thème Compilateur** ainsi qu'au manuel Mode Développement de 4D.

Langue des commandes et des constantes

A compter de 4D v15, l'éditeur de méthodes de 4D utilise par défaut le mode international "Anglais-US", quelle que soit la langue de l'application 4D ou les paramètres système locaux. Cette fonctionnalité rend le code indépendant de toute variation régionale susceptible de perturber son interprétation lorsque différentes versions de l'application 4D sont utilisées (formats de date par exemple) ; en outre, dans la version française de 4D, les commandes et les constantes sont exprimées en "anglais-US".

Ce paramétrage par défaut apporte plusieurs bénéfices aux développeurs 4D :

- Il facilite le partage du code entre les développeurs, quels que soient leur pays, paramètres régionaux, ou version de 4D. Une méthode 4D peut être échangée par simple copier/coller ou enregistrée dans un fichier texte, sans aucun problème de compatibilité.
- Il rend également les méthodes 4D compatibles avec l'utilisation d'outils de contrôle des sources, qui requièrent souvent que les exports soient indépendants des paramètres régionaux ou des langues.

Ce paramétrage peut être désactivé via l'option "Utiliser langage français et paramètres régionaux système" de la boîte de dialogue des Préférences de 4D (cf. **Page Méthodes**).

Principes de saisie en langage Anglais-US

Le mode de langage utilisé influence la façon d'écrire des méthodes. Cela concerne le code en mode développement ainsi que les formules saisies dans les applications finales. Dans le mode par défaut (Anglais-US), les règles suivantes s'appliquent :

- le séparateur décimal pour les nombres réels est le point (".") dans toutes les versions (et non la virgule (",") comme c'est l'usage en français par exemple).
- les constantes de type date doivent utiliser le format ISO (!YYYY-MM-DD!) dans toutes les versions.
- les noms des commandes et des constantes sont en anglais (cette modification ne concerne que la version française de 4D car le langage est déjà en anglais dans toutes les autres langues de 4D).

Note : L'éditeur de méthodes inclut des mécanismes spécifiques permettant de corriger automatiquement les saisies incorrectes si nécessaire.

Le tableau suivant illustre les différences de code entre 4D v15 (ou suivantes) et les versions précédentes :

	Exemples de code dans les méthodes ou formules
4D v15 et suivantes (Mode par défaut, toutes langues)	a:=12.50 b:=!2013-12-31! Current date
4D v14 ou 4D v15 (préférence cochée, version US par exemple)	a:=12.50 b:=!12/31/2013! Current date
4D v14 ou 4D v15 (préférence cochée, version française)	a:=12,50 b:=!31/12/2013! Date du jour

Note : Lorsque la préférence est cochée, les formats de date et de réel sont basés sur les paramètres système.

Types de données

De nombreux types de données peuvent être stockés dans une base 4D. Le langage distingue huit types de données élémentaires : chaîne, numérique, date, heure, booléen, image, pointeur et objet.

- **Chaîne** : Une suite de caractères, telles que “Bonjour à tous”. Les champs et variables texte ainsi que les champs alpha sont des données de type Chaîne.
- **Numérique** : Nombres, tels que 2 ou 1 000,67. Les champs et variables entier, entier long et numérique (aussi appelé réel) sont des données de type Numérique.
- **Date** : Dates telles que 20/11/2014. Les champs et variables date sont des données de type Date.
- **Heures** : Heures, c’est-à-dire heures, minutes et secondes, telles que 21:00:00 ou 4:35:30 de l'après-midi. Les champs et variables heure sont des données de type Heure.
- **Booléen** : Valeurs logiques de VRAI ou FAUX. Les champs et variables booléens sont des données de type Booléen.
- **Image** : Les champs et variables image sont des données de type Image.
- **Pointeur** : Type spécial de données, utilisé en programmation avancée. Les variables pointeur sont des données de type Pointeur. Il n’y a pas de type de champ correspondant.
- **Objet** : Type de données composite pouvant contenir des ensemble de données de tout type sous forme de paires clés/valeurs. Les champs et variables objet sont des données de type Objet.

Vous constatez que, dans cette liste de types de données, les types chaîne et numérique sont associés à plus d’un type de champ. Lorsque des données sont placées dans un champ, le langage les convertit automatiquement dans le type du champ. Par exemple, si un champ de type entier est utilisé, les valeurs qu’il contient sont automatiquement traitées en tant que numériques. En d’autres termes, vous n’avez pas à vous préoccuper du mélange de champs de types semblables lorsque vous programmez avec 4D ; le langage le gère pour vous.

Cependant, il est important, lorsque vous utilisez le langage, de ne pas mélanger les types de données différents. Tout comme il est absurde de stocker la valeur “ABC” dans un champ de type Date, il est absurde de donner la valeur “ABC” à une variable utilisée pour des dates. Dans la plupart des cas, 4D est très tolérant et tentera d’utiliser de manière logique ce que vous faites. Par exemple, si vous additionnez un nombre x et une date, 4D déduira que vous voulez ajouter x jours à la date, mais si vous tentez d’ajouter une chaîne à une date, 4D vous préviendra que cette opération est impossible.

Certains cas nécessitent que vous stockiez des données dans un type et que vous les utilisiez dans un autre. Le langage contient un ensemble complet de commandes vous permettant de convertir des types de données vers d’autres types. Par exemple, si vous voulez créer un numéro de matricule commençant par des chiffres et se terminant par des lettres, vous pouvez écrire :

```
[Produits]Matricule:=Chaine(Numéro)+"abc"
```

où, si **Numéro** vaut 17, *[Produits]Matricule* prendra la valeur “17abc”.

Les types de données sont détaillés dans la section **Types de données**.

Opérateurs

Lorsque vous programmez avec 4D, il est rare que vous ayez simplement besoin de données “brutes”. Le plus souvent, il sera nécessaire de traiter ces données d'une manière ou d'une autre. Vous effectuez ces calculs avec des **opérateurs**. Les opérateurs, en général, prennent deux valeurs et effectuent avec elles une opération dont le résultat est une troisième valeur. Vous connaissez déjà la plupart des opérateurs. Par exemple, 1 + 2 utilise l’opérateur d’addition (ou signe plus) pour faire la somme de deux nombres, et le résultat est 3. Le tableau ci-dessous présente quelques opérateurs courants :

Opérateur	Opération	Exemple
+	Addition	1 + 2 ce qui fait 3
–	Soustraction	3 – 2 ce qui fait 1
*	Multiplication	2 * 3 ce qui fait 6
/	Division	6 / 2 ce qui fait 3

Les opérateurs numériques ne représentent qu’un seul des différents types d’opérateurs disponibles. Comme 4D traite de multiples types de données, tels que des nombres, des dates ou des images, vous disposez d’opérateurs particuliers effectuant des opérations sur ces données.

Souvent, les mêmes symboles sont utilisés pour des opérations différentes, en fonction du type de données traitées. Par exemple, le signe (+) peut effectuer diverses opérations, comme le montre le tableau suivant :

Type de données	Opération	Exemple
Numérique	Addition	1 + 2 ajoute les nombres, le résultat est 3
Chaîne	Concaténation	“Bonjour ” + “à tous” concatène (met bout à bout) les chaînes, le résultat est “Bonjour à tous”
Date et Numérique	Addition de date	!1997-01-01! + 20 ajoute 20 jours à la date 1 janvier 1997, le résultat est la date 21 janvier 1997

Les opérateurs sont détaillés dans la section **Opérateurs** et ses sous-sections.

Expressions

Pour parler simplement, les expressions retournent une valeur. En fait, lorsque vous programmez avec 4D, vous utilisez tout le temps des expressions et vous avez tendance à les manipuler uniquement à travers la valeur qu’elles représentent. Les expressions sont aussi appelées formules.

Les expressions peuvent être constituées de presque tous les éléments du langage : commandes, opérateurs, variables et champs. Vous utilisez des expressions pour écrire des lignes de code, qui sont à leur tour utilisées pour construire des méthodes. Des expressions sont employées à chaque fois que le langage de 4D a besoin de connaître la valeur d’une donnée.

Les expressions sont rarement “indépendantes”. Il n’y a que peu d’endroits dans 4D où une expression peut être utilisée en tant que telle : dans la boîte de dialogue de Recherche par formule, dans la fenêtre du Débogueur où la valeur des expressions peut être évaluée, dans la boîte de dialogue Appliquer une formule, et dans l’éditeur d’états semi-automatiques en tant que formule dans une colonne.

Une expression peut être simplement une constante, telle que le chiffre 4 ou la chaîne “Bonjour”. Comme son nom l’indique, la valeur d’une constante ne change jamais.
C’est lorsqu’elles contiennent des opérateurs que les expressions commencent à devenir intéressantes. Dans les paragraphes précédents, vous avez déjà pu voir des expressions utilisant des opérateurs. Par exemple, 4 + 2 est une expression qui utilise l’opérateur d’addition pour additionner deux nombres, et dont le résultat est 6.

Vous vous référez à une expression par le biais du type de données qu’elle retourne. Il existe huit types d’expressions :

- Expression chaîne,
- Expression numérique (aussi appelée nombre),
- Expression date,

- Expression heure,
- Expression booléenne,
- Expression image,
- Expression pointeur,
- Expression objet.

Le tableau suivant fournit un exemple pour chaque type d'expression.

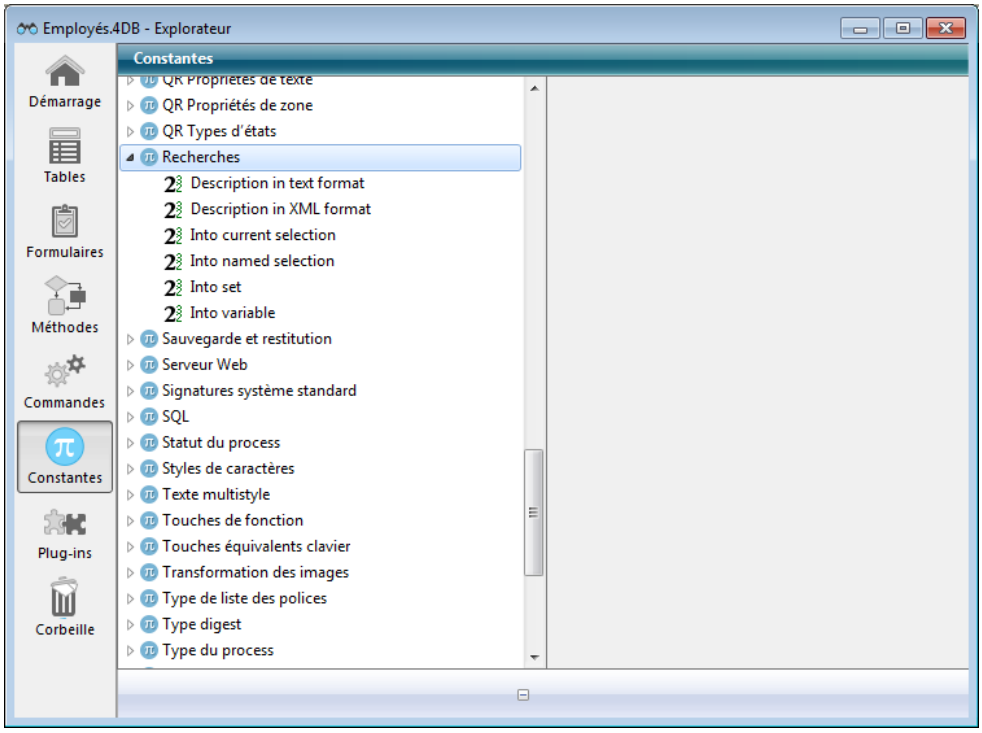
Expression	Type	Description
"Bonjour"	Chaîne	Le mot Bonjour est une constante chaîne, signalée par les guillemets
"Bonjour " + "à tous"	Chaîne	Deux chaînes, "Bonjour " et "à tous", sont mises bout à bout (concaténées) à l'aide de l'opérateur de concaténation de chaînes (+). La chaîne "Bonjour à tous" est retournée.
"M. " + [Amis]Nom	Chaîne	Deux chaînes sont concaténées : la chaîne "M. " et la valeur courante du champ Nom de la table Amis. Si le champ contient "Dupont", l'expression retourne "M. Dupont".
Majusc("dupont")	Chaîne	Cette expression utilise "**Majusc**", une commande du langage, pour convertir la chaîne "dupont" en majuscules. Elle retourne "DUPONT".
4	Numérique	C'est une constante numérique, 4.
4 * 2	Numérique	Deux nombres, 4 et 2, sont multipliés à l'aide de l'opérateur de multiplication (*). Le résultat est le nombre 8.
MonBouton	Numérique	C'est le nom d'un bouton. Il retourne la valeur courante du bouton : 1 s'il y a eu un clic sur le bouton, 0 sinon.
!1997-01-25!	Date	C'est une constante date pour la date 25/01/97 (25 janvier 1997). Notez l'emploi des points d'exclamation pour indiquer une constante date.
Date du jour + 30	Date	C'est une expression de type Date qui utilise la fonction **Date du jour** pour récupérer la date courante. Elle ajoute 30 jours à la date d'aujourd'hui et retourne la nouvelle date.
?8:05:30?	Heure	C'est une constante heure qui représente 8 heures, 5 minutes, et 30 secondes.
?2:03:04? + ? 1:02:03?	Heure	Cette expression ajoute une heure à une autre et retourne l'heure 3:05:07.
Vrai	Booléen	Cette commande retourne la valeur booléenne VRAI.
10 # 20	Booléen	C'est une comparaison logique entre deux nombres. Le symbole (#) signifie "est différent de". Comme 10 "est différent de" 20, l'expression retourne VRAI.
"ABC" = "XYZ"	Booléen	C'est une comparaison logique entre deux chaînes. Elles sont différentes, donc l'expression retourne FAUX.
MonImage + 50	Image	Cette expression considère l'image placée dans MonImage, la déplace de 50 pixels vers la droite, et retourne l'image résultante.
->[Amis]Nom	Pointeur	Cette expression retourne un pointeur vers lechamp [Amis]Nom.
Table (1)	Pointeur	C'est une commande qui retourne un pointeur vers la première table.
JSON Parse(MaChaine)	Objet	C'est une commande qui retourne MaChaine sous forme d'objet (si format adéquat)

Constantes

Une constante est une expression dont la valeur est fixe. Il existe deux types de constantes : les **constantes prédéfinies** que vous pouvez appeler en inscrivant leur nom et les **constantes littérales**, pour lesquelles vous devez saisir une valeur.

Constantes prédéfinies

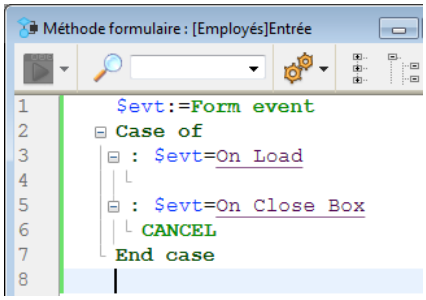
4D propose un ensemble de **constantes prédéfinies**. Ces constantes sont regroupées par thèmes dans la fenêtre de l'Explorateur :



Pour utiliser une constante prédéfinie dans la fenêtre de l'éditeur de méthodes, vous pouvez :

- soit glisser-déposer la constante depuis la fenêtre de l'Explorateur vers la fenêtre de l'éditeur de méthodes,
- soit saisir directement son nom dans la fenêtre de l'éditeur de méthodes. La fonction de saisie prédictive propose les constantes correspondant au contexte de programmation.

Les constantes prédéfinies apparaissent soulignées par défaut dans la fenêtre de l'éditeur de méthodes et dans la fenêtre du débogueur :



Dans la fenêtre ci-dessus, On Load est par exemple une constante prédéfinie.

Constantes littérales

4D accepte quatre types de données pour les constantes littérales :

- Chaîne,
- Numérique,
- Date,
- Heure.

Constantes chaînes

Une constante de type chaîne est incluse entre des guillemets droits ("…"). Voici quelques exemples de constantes chaîne :
"Ajouter Enregistrements"
"Aucun enregistrement trouvé."
"Facture"

Une chaîne vide est spécifiée par la succession de deux guillemets ("").

Constantes numériques

Une constante numérique s'écrit comme un nombre réel. Voici quelques exemples de constantes numériques :
27
123.76
0.0076

Les nombres négatifs s'écrivent précédés du signe moins (-). Par exemple:
-27
-123.76
-0.0076

Note : Depuis 4D v15, le séparateur décimal est par défaut le point (.), quelle que soit la langue du système. Si vous avez coché l'option "Utiliser langage français et paramètres régionaux système" (cf. **Page Méthodes**), vous devez utiliser le séparateur défini dans votre système.

Constantes dates

Une constante de type date est incluse entre deux points d'exclamation (!…!). Depuis 4D v15, une date doit être structurée avec le format ISO (!YYYY-MM-DD!). Voici quelques exemples de constantes dates :
!1976-01-01!
!2004-09-29!
!2015-12-31!

Une date nulle s'écrit !00-00-00!

Astuce : L'éditeur de méthodes dispose d'un raccourci pour entrer une date nulle. Pour cela, tapez un point d'exclamation (!) et appuyez sur la touche **Entrée**.

Notes :

- Pour des raisons de compatibilité, 4D accepte que l'année soit saisie sur deux chiffres. Dans ce cas, le programme considère qu'elle appartient au XXe ou au XXIe siècle selon qu'elle est supérieure ou inférieure à 30, sauf si ce fonctionnement par défaut a été modifié à l'aide de la commande **SIECLE PAR DEFAUT**.
- Si vous avez coché l'option "Utiliser langage français et paramètres régionaux système" (cf. **Page Méthodes**), vous devez utiliser le format de date défini dans votre système. Généralement dans un environnement français, une date est saisie sous la forme jour/mois/année, une barre oblique "/" séparant les valeurs.

Constantes heures

Une constante heure est incluse entre deux points d'interrogation (?...?).

Avec une version française de 4D, une heure est structurée sous la forme heure:minute:seconde, deux points (:) séparant les valeurs. Les heures sont stockées dans un format de 24 heures.

Voici quelques exemples de constantes heures :
?00:00:00? ` minuit
?09:30:00? ` 9:30 du matin
?13:01:59? ` 13 heures, 1 minute et 59 secondes

Une heure nulle s'écrit ?00:00:00?.

Astuce : L'éditeur de méthodes dispose d'un raccourci pour saisir une heure nulle. Pour cela, tapez un point d'interrogation (?) et appuyez sur la touche Entrée.

Variables

Fondamentalement, dans 4D, les données peuvent être stockées de deux manières. Les champs stockent les données sur disque, de manière permanente ; les variables stockent les données en mémoire, de manière temporaire.

Lorsque vous définissez votre base, vous indiquez à 4D les noms et les types de champs que vous voulez utiliser. C'est pratiquement la même chose pour les variables — vous leur donnez un nom et un type.

Les types de variables suivants correspondent à chacun des types de données :

- Alpha(*) ou Texte : chaîne alphanumérique pouvant contenir jusqu'à 2 Go de texte
- Entier : nombre entier compris entre -32768 et 32767
- Entier long : nombre entier compris entre -2^31 et (2^31) -1
- Réel (ou Numérique) : nombre réel compris entre ±1.7e±308 (13 chiffres significatifs)
- Date : date comprise entre 1/1/100 et 31/12/32767
- Heure : heure comprise entre 00:00:00 et 596000:00:00 (secondes depuis minuit)
- Booléen : Vrai ou Faux
- Image : toute image Windows ou Mac OS
- Objet : ensemble de paires "propriété/valeur" structurées dans un format de type JSON
- BLOB (Binary Large OBject) : suite d'octets pouvant aller jusqu'à 2 Go
- Pointeur : pointeur vers une table, un champ, une variable, un tableau ou un élément de tableau.

(*) En mode Unicode, les types de variables Alpha et Texte sont identiques. En mode non Unicode (mode compatibilité), un Alpha est une chaîne alphanumérique fixe pouvant comprendre jusqu'à 255 caractères.

Vous pouvez afficher des variables à l'écran (à l'exception des pointeurs et des BLOB), les utiliser pour saisir des données, et les imprimer dans des états. Dans ces cas, elles se comportent exactement comme des champs, et les mêmes contrôles intégrés sont disponibles lorsque vous les créez :

- Formats d'affichage,
- Contrôles de saisie tels que filtres de saisie et valeurs par défaut,
- Filtrages des caractères,
- Enumérations (listes hiérarchiques)
- Valeurs saisissables ou non-saisissables

Les variables peuvent également servir à :

- contrôler des boutons (boutons, cases à cocher, boutons radio, boutons 3D, etc.),
- contrôler des thermomètres, règles et cadrans,
- contrôler des zones de défilement, des pop up menus et des listes déroulantes,
- contrôler des listes hiérarchiques et des menus hiérarchiques,
- contrôler des grilles de boutons, onglets, boutons image, etc.
- afficher les résultats de calculs ne devant pas être sauvegardés.

Créer des variables

Vous pouvez créer des variables simplement en les utilisant ; il n'est pas obligatoire de les déclarer formellement comme vous le faites avec les champs. Par exemple, si vous voulez créer une variable qui contient la date du jour plus 30 jours, il vous suffit d'écrire dans 4D :

```
MaDate:=Date du jour+30
```

MaDate est alors créée et contient la valeur que vous voulez. Le programme interprète la ligne comme "MaDate prend la valeur de la date courante plus 30 jours". Vous pourrez utiliser *MaDate* à chaque fois que vous le souhaiterez dans votre base. Par exemple, vous pouvez la stocker dans un champ du même type :

```
[MaTable]MonChamp:=MaDate
```

Toutefois, il est généralement conseillé de définir explicitement le type d'une variable. Pour plus d'informations sur le typage des variables dans une base, reportez-vous à la section **Compilateur**.

Assigner des valeurs aux variables

Vous pouvez donner des valeurs aux variables et/ou récupérer leur valeur. Donner une valeur à une variable s'appelle **assigner une valeur** (ou **affecter une valeur**) et s'effectue à l'aide de l'opérateur d'assignation (:=). L'opérateur d'assignation est également utilisé pour assigner des valeurs aux champs.

L'opérateur d'assignation est le premier moyen pour créer une variable et lui donner une valeur. Vous placez le nom de la variable que vous voulez créer à gauche de l'opérateur. Par exemple :

```
MonNombre:=3
```

crée la variable *MonNombre* et lui donne la valeur numérique 3. Si *MonNombre* existait déjà, elle prend simplement la valeur 3.

Bien entendu, les variables ne seraient pas très utiles si vous ne pouviez pas récupérer les valeurs qu'elles contiennent. De nouveau, vous utilisez l'opérateur d'assignation. Si vous devez placer la valeur de *MonNombre* dans un champ nommé *[Produits]Taille*, il vous suffit de placer *MonNombre* à droite de l'opérateur d'assignation :

```
[Produits]Taille:=MonNombre
```

Dans ce cas, *[Produits]Taille* vaudrait 3. Cet exemple est plutôt simple, mais il illustre le moyen élémentaire dont vous disposez pour transférer des données d'un objet vers un autre en utilisant le langage.

Important : Ne confondez pas l'opérateur d'assignation (:=) avec le signe égal (=) qui est un opérateur de comparaison. L'assignation et la comparaison sont deux opérations très différentes. Pour plus d'informations sur les opérateurs de comparaison, reportez-vous à la section **Opérateurs**.

Variables locales, process et interprocess

Vous pouvez créer trois types de variables : des variables **locales**, des variables **process** et des variables **interprocess**. La différence entre ces trois types de variables est leur portée, ou les objets pour lesquels elles sont disponibles.

Variables locales

Le premier type de variable est la variable locale. Une variable locale, comme son nom l'indique, est locale à une méthode — c'est-à-dire accessible uniquement à l'intérieur de la méthode dans laquelle elle a été créée et inaccessible à l'extérieur de cette méthode. Pour une variable, être locale à une méthode signifie avoir une portée locale.

Vous utilisez une variable locale lorsque vous souhaitez limiter son fonctionnement à la méthode, pour une des raisons suivantes :

- Eviter des conflits de noms avec les autres variables.
- Utiliser temporairement des valeurs,
- Réduire le nombre de variables process.

Le nom d'une variable locale commence toujours par le signe dollar ($) et peut contenir jusqu'à 31 autres caractères. Si vous saisissez un nom plus long, 4D le tronque pour le ramener à 31 caractères.

Lorsque vous développez une base comportant de nombreuses méthodes et variables, il arrive souvent que vous n'ayez besoin d'utiliser une variable que dans une méthode. Vous pouvez alors créer et utiliser une variable locale, sans devoir vous soucier de l'existence d'une autre variable du même nom ailleurs dans la base.

Fréquemment, dans une base de données, des informations ponctuelles sont demandées à l'utilisateur. La commande **Demander** peut être appelée pour obtenir ces informations. Elle affiche une boîte de dialogue comportant un message demandant à l'utilisateur de répondre et, lorsque la réponse est validée, la retourne. Généralement, il n'est pas nécessaire de conserver cette information très longtemps dans vos méthodes. C'est l'endroit parfait pour utiliser une variable locale. Voici un exemple :

```
$vsID:=Demander("Saisissez votre numéro d'identification :")
Si(OK=1)
   CHERCHER([Personnes];[Personnes]ID=$vsID)
Fin de si
```

Cette méthode demande simplement à l'utilisateur de saisir un numéro d'identification. La réponse est placée dans une variable locale, *$vsID*, puis la méthode la recherche parmi les champs *[Personnes]ID*. Une fois la méthode terminée, la variable locale *$vsID* est effacée de la mémoire. Ce fonctionnement est bien adapté puisque la variable n'est utile qu'une seule fois et dans cette méthode uniquement.

Variables process

Le second type de variable est la variable process. Une variable process est "visible" uniquement dans le process où elle a été créée. Elle est utilisable par toutes les méthodes du process et toutes les méthodes appelées depuis le process.

Le nom d'une variable process ne comporte aucun préfixe. Une variable process peut comporter jusqu'à 31 caractères.

En mode interprété, les variables sont gérées dynamiquement : elles sont créées en mémoire et effacées "à la volée". En mode compilé, tous les process que vous créez (process utilisateurs) partagent la même définition des variables process, mais chaque process dispose de sa propre instance pour chaque variable. Par exemple, la variable *maVar* est une certaine variable dans le process **P_1** et une autre variable dans le process **P_2**.

Un process peut lire et écrire des variables process dans un autre process à l'aide des commandes **LIRE VARIABLE PROCESS** et **ECRIRE VARIABLE PROCESS**. Nous vous recommandons de n'utiliser ces commandes que dans le cadre des besoins décrits ci-dessous (qui sont les raisons pour lesquelles ces commandes ont été créées dans 4D) :

- Communication interprocess à des endroits particuliers de votre code
- Gestion du glisser-déposer interprocess
- En client/serveur, communication entre les process sur les postes clients et les procédures stockées exécutées sur le serveur.

Pour plus d'informations, reportez-vous à la section **Process** et à la description de ces commandes.

Variables interprocess

Le troisième type de variable est la variable interprocess. Les variables interprocess sont visibles dans toute la base et sont disponibles pour tous les process.

Le nom d'une variable interprocess débute toujours par le symbole (<>) — formé du symbole "inférieur à" suivi du symbole "supérieur à" — et peut comporter jusqu'à 31 caractères supplémentaires.

Note : Cette syntaxe peut être utilisée sur les plates-formes Windows et Macintosh. En outre, sous Mac OS uniquement, vous pouvez utiliser le symbole "diamant" (Option+v sur un clavier français).

Les variables interprocess sont principalement utilisées pour le partage d'informations entre les process.

En mode client/serveur, chaque poste (client et serveur) partage la même définition des variables interprocess, mais chacun utilise une instance différente d'une variable.

Variables objets dans les formulaires

Dans l'éditeur de formulaires, le fait de donner un nom à un objet actif — bouton, bouton radio, case à cocher, zone de défilement, thermomètre, etc. — crée automatiquement une variable, ayant par défaut le même nom. Par exemple, si vous créez un bouton appelé *MonBouton*, une variable *MonBouton* est également créée. Notez bien que ce nom de variable n'est pas le libellé du bouton, mais son nom.

Ces variables vous permettent de contrôler et de gérer vos objets. Par exemple, lorsque l'utilisateur clique sur un bouton, la variable du bouton prend la valeur 1 ; sinon, le reste du temps, elle est à 0. La variable associée à un thermomètre ou un cadran vous permet de lire et de modifier les valeurs représentées. Par exemple, si l'utilisateur clique dans un thermomètre pour fixer une nouvelle valeur, la valeur de la variable est modifiée en conséquence. De même, si une méthode change la valeur de la variable, le thermomètre est redessiné pour afficher cette nouvelle valeur.

Pour plus d'informations sur les variables et les formulaires, reportez-vous au manuel Mode Développement de 4D ainsi qu'à la description de la fonction **Evénements formulaire**.

Variables dynamiques

Vous pouvez laisser à 4D le soin de créer dynamiquement et en fonction des besoins, les variables associées à vos objets de formulaires (boutons, variables saisissables, cases à cocher, etc.). Pour cela, il suffit de laisser vide le champ "Nom de la variable" dans la Liste des propriétés pour l'objet :

Lorsqu'une variable n'est pas nommée, au moment du chargement du formulaire, 4D crée pour l'objet une nouvelle variable avec un nom calculé unique dans l'espace des variables process de l'interpréteur (ce qui permet d'utiliser ce mécanisme même en mode compilé). Cette variable temporaire sera détruite à la fermeture du formulaire.

Pour que ce principe puisse fonctionner en mode compilé, il est impératif que les variables dynamiques soient explicitement typées. Pour cela, vous disposez de deux possibilités :

- définir le type via le menu "" de la Liste des propriétés.
  Note : Lorsque la variable est nommée, le menu "Type de variable" ne type pas réellement la variable mais permet uniquement de mettre à jour les options de la Liste des propriétés (hormis pour les variables image). Pour typer une variable nommée, il est nécessaire d'utiliser les commandes du thème **Compilateur**.
- utiliser un code d'initialisation spécifique lors du chargement du formulaire, en utilisant par exemple la commande **VARIABLE VERS VARIABLE** :

```
Si(Evenement formulaire=Sur chargement)
   C_TEXTE($init)
   $Ptr_object:=OBJET Lire pointeur(Objet nommé;"Commentaires")
   $init:=""
   VARIABLE VERS VARIABLE(Numero du process courant;$Ptr_object->;$init)
Fin de si
```

Note : Si vous définissez une variable dynamique, sélectionnez la valeur **Aucun** dans le menu "Type de variable" et n'utilisez pas de code d'initialisation, une erreur de typage sera retournée par le compilateur.

Dans le code 4D, les variables dynamiques sont accessibles via un pointeur obtenu avec la commande **OBJET Lire pointeur**. Par exemple :

```
 // affecter l'heure 12:00:00 à la variable de l'objet "hdebut"
$p :=OBJET Lire pointeur(Objet nommé;"hdebut")
$p->:=?12:00:00?
```

L'intérêt de ce mécanisme est double :

- Il permet le développement de composants de type "sous-formulaire" pouvant être utilisés plusieurs fois dans un même formulaire hôte. Prenons le cas par exemple d'un sous-formulaire *datepicker* inséré deux fois dans un formulaire hôte pour définir une date de début et une date de fin. Ce sous-formulaire va utiliser des objets pour le choix du jour du mois et de l'année. Il faut donc que ces objets travaillent avec des variables différentes pour la date de début et la date de fin. Laisser 4D créer leur variable avec un nom unique permet de résoudre cette difficulté.
- Il permet de limiter la consommation mémoire. En effet, les objets de formulaire ne travaillent qu'avec des variables process ou interprocess. Or, en mode compilé, une instance de chaque variable process est créée dans tous les process, y compris les process du serveur. Cette instance consomme de la mémoire, même si le formulaire n'est pas utilisé au cours de la session. Laisser 4D créer dynamiquement les variables au chargement des formulaires permet donc d'économiser la mémoire.

Note : Lorsqu'il n'y a pas de nom de variable, c'est le nom de l'objet encadré de guillemets qui est affiché dans l'éditeur de formulaires (lorsque l'objet affiche par défaut un nom de variable).

Variables système

4D exploite un certain nombre de variables particulières appelées **variables système**. Ces variables vous permettent de contrôler de nombreuses opérations. Toutes les variables système sont des variables process, disponibles à l'intérieur d'un seul process.

La plus importante de toutes est la variable système **OK**. Comme son nom le laisse supposer, elle vous indique si tout est OK dans un process particulier. Est-ce que l'enregistrement a bien été sauvegardé ? Est-ce que l'import d'enregistrements s'est bien déroulé ? Est-ce que l'utilisateur a cliqué sur le bouton OK ? La variable système OK prend la valeur 1 lorsqu'une opération s'est correctement déroulée, et 0 dans le cas contraire.

Pour plus d'informations sur les variables système, reportez-vous à la section traitant des **Variables système**.

Variables système

4D gère un certain nombre de variables appelées **variables système**. Ces variables vous permettent de contrôler le déroulement de diverses opérations. Les variables système sont toutes des variables process, accessibles uniquement à l'intérieur d'un process. Cette section décrit les variables système de 4D.

Pour plus d'informations sur le type de ces variables, reportez-vous au paragraphe **Variables système** dans la section **Guide du typage**.

OK

La variable système **OK** est la plus couramment utilisée. En général, elle prend la valeur 1 lorsqu'une opération s'est correctement déroulée, et 0 lorsque l'opération a échoué.

De nombreuses commandes 4D modifient la valeur de la variable système **OK**. Reportez-vous à la description de chaque commande pour savoir si elle met à jour cette variable système.

Dans cette documentation, le pictogramme  indique qu'une commande modifie la valeur de la variable OK. Vous pouvez cliquer sur cette image pour générer la liste de toutes les commandes concernées.

Document

La variable système **Document** contient le chemin d'accès et le nom du dernier fichier disque ayant été ouvert ou créé à l'aide d'une des commandes suivantes :

Ajouter a document	CHARGER ENSEMBLE
Creer document	_o_Créer fichier ressources
ECRIRE FICHIER IMAGE	ECRIRE VARIABLES
EXPORTER TEXTE	ECRITURE DIF
ECRITURE SYLK	QR ETAT
EXPORTER DONNEES	FIXER FICHIER HISTORIQUE
GENERER APPLICATION	IMPORTER DONNEES
IMPRIMER ETIQUETTES	IMPORTER TEXTE
LECTURE DIF	LECTURE SYLK
LIRE FICHIER IMAGE	LIRE ICONE DOCUMENT
LIRE VARIABLES	Ouvrir document
Ouvrir fichier ressources	Selectionner document
STOCKER ENSEMBLE	REGLER SERIE
UTILISER FILTRE	

FldDelimit

La variable système **FldDelimit** contient le code du caractère à utiliser comme délimiteur de champs lorsque vous importez ou exportez du texte. Par défaut, cette valeur est 9, c'est-à-dire le code du caractère Tabulation. Modifiez cette valeur pour changer de délimiteur de champs.

RecDelimit

La variable système **RecDelimit** contient le code du caractère à utiliser comme délimiteur d'enregistrements lorsque vous importez ou exportez du texte. Par défaut, cette valeur est 13, c'est-à-dire le code du caractère Retour chariot. Modifiez cette valeur pour changer de délimiteur d'enregistrements.

Error, Error method, Error line, Error formula

Ces variables ne sont utilisables que dans une méthode d'interception d'erreurs installée par la commande **APPELER SUR ERREUR**. Si vous souhaitez qu'elles soient accessibles dans la méthode ayant provoqué l'erreur, copiez leur valeur dans vos propres variables process.

- **Error** : Variable système de type entier long. Cette variable contient le code de l'erreur. Les codes des erreurs de 4D et des erreurs Système sont listés dans les sections du thème **Codes d'erreurs**.
- **Error method** : Variable système de type texte. Cette variable contient le nom complet de la méthode ayant déclenché l'erreur.
- **Error line** : Variable système de type entier long. Cette variable contient le numéro de la ligne à l'origine de l'erreur dans la méthode ayant déclenché l'erreur.
- **Error formula** : Variable système de type texte. Cette variable contient la formule de code 4D (en texte brut) qui est à l'origine de l'erreur. Le texte de la formule est exprimé dans la langue courante du langage de 4D.
  Si le code source responsable de l'erreur ne peut pas être trouvé, **Error formula** contient une chaîne vide. Ce cas peut se produire dans les bases compilées lorsque :
  - le code source a été supprimé de la structure compilée à l'aide du générateur d'application.
  - le code source est disponible mais la base a été compilée sans l'option **Contrôle d'exécution**.

MouseDown, MouseX, MouseY, KeyCode, Modifiers et MouseProc

Ces variables système ne sont utilisables que dans une méthode installée par **APPELER SUR EVENEMENT** (exceptées **MouseX** et **MouseY** dans certains cas, voir ci-dessous).

- La variable système **MouseDown** prend la valeur 1 si le bouton de la souris a été enfoncé. Sinon, elle prend la valeur 0.
- Si l'événement est un **MouseDown** (**MouseDown**=1), les variables système **MouseX** et **MouseY** contiennent les coordonnées verticale et horizontale de l'endroit où le clic a eu lieu. Les deux valeurs sont exprimées en pixels et avec le système de coordonnées locales de la fenêtre.
- Dans le contexte d'un clic sur un champ image ou une variable image, les variables système **MouseX** et **MouseY** retournent les coordonnées locales du clic dans les événements formulaire Sur clic, Sur double clic et Sur relâchement bouton. Les coordonnées locales du clic souris sont également retournées avec les événements formulaire Sur début survol et Sur survol. Les coordonnées sont exprimées en pixels en partant du coin supérieur gauche de l'image (0,0) Pour plus d'informations, reportez-vous à la section **Introduction aux images** et à la commande **SVG Chercher ID element par coordonnees**.

- La variable système **KeyCode** contient le code de la touche ayant été enfoncée. Si la touche enfoncée était une touche de fonction, **KeyCode** contient un code spécial. Les codes de caractères et les codes des touches de fonction sont listés dans les sections **Codes Unicode**, **EXPORTER TEXTE** et **Codes des touches de fonction**.
- La variable système **Modifiers** contient les codes des modifiers du clavier (**Ctrl/Commande**, **Alt/Option**, **Maj**, **Verr. Maj**). Cette variable n'est significative que dans une méthode d'interruption sur événement installée par la commande **APPELER SUR EVENEMENT**.
- La variable système **MouseProc** contient le numéro du process dans lequel le dernier événement a eu lieu.

Pointeurs

Description

Les pointeurs sont des outils de programmation avancée.

Lorsque vous utilisez le langage de 4D, vous vous référez aux différents objets par l'intermédiaire de leur nom — en particulier les tables, champs, variables et tableaux. Pour appeler l'un d'entre eux, vous écrivez simplement son nom. Cependant, il est parfois utile de pouvoir appeler ou référencer ces éléments sans nécessairement connaître leur nom. C'est ce que permettent les pointeurs.

Le concept de pointeur n'est pas tellement éloigné de la vie courante. Vous vous référez souvent à des choses sans connaître leur identité exacte. Par exemple, vous dites à un ami "Allons-y avec ta voiture" au lieu de "Allons-y avec la voiture immatriculée 123 ABD 99". Dans ce cas, vous faites référence à la voiture immatriculée 123 ABD 99 en utilisant l'expression "ta voiture". Par analogie, l'expression "la voiture immatriculée 123 ABD 99" est le nom d'un objet, et "ta voiture" est un pointeur référençant (ou pointant vers) l'objet.

La capacité de se référer à quelque chose sans connaître son identité exacte est très utile. Si votre ami s'achetait une nouvelle voiture, l'expression "ta voiture" serait toujours exacte — ce serait toujours une voiture et vous pourriez toujours aller quelque part avec. Les pointeurs fonctionnent de la même manière. Par exemple, un pointeur peut pointer à un moment donné vers un champ numérique appelé Age, et plus tard vers une variable numérique appelée Ancien âge. Dans les deux cas, le pointeur référence des données numériques pouvant être utilisée dans des calculs. Vous pouvez utiliser des pointeurs pour référencer des tables, des champs, des variables, des tableaux et des éléments de tableaux.

Le tableau suivant vous fournit un exemple de chaque type :

Objet	Référencement	Utilisation	Affectation
Table	vpTble:=->[Table]	TABLE DEFAUT(vpTble->)	n/a
Champ	vpChp:=->[Table]Chp	ALERTE(vpChp->)	vpChp->:="Jean"
Variable	vpVar:=->Variable	ALERTE(vpVar->)	vpVar->:="Jean"
Tableau	vpT:=->Tableau	TRIER TABLEAU(vpT->;>)	COPIER TABLEAU(Tab;vpT->)
Elém. tabl.	vpElem:=->Tableau{1}	ALERTE(vpElem->)	vpElem->:="Jean"

Utiliser des pointeurs : un exemple

Il est plus facile d'expliquer l'utilisation des pointeurs au travers d'un exemple. Cet exemple vous montre comment accéder à une variable par l'intermédiaire d'un pointeur. Nous commençons par créer la variable :

```
MaVar:="Bonjour"
```

MaVar est désormais une variable contenant la chaîne "**Bonjour**". Nous pouvons alors créer un pointeur vers **MaVar** :

```
MonPointeur:=->MaVar
```

Le symbole -> signifie "pointer vers" (ce symbole est formé du caractère "tiret" (-) suivi du caractère "supérieur à"). Dans ce cas, il crée un pointeur qui référence ou "pointe vers" **MaVar**. Ce pointeur est assigné à **MonPointeur** via l'opérateur d'assignation.

MonPointeur est désormais une variable qui contient un pointeur vers **MaVar**. **MonPointeur** ne contient pas "**Bonjour**", la valeur de **MaVar**, mais vous pouvez utiliser **MonPointeur** pour obtenir la valeur contenue dans **MaVar**. L'expression suivante retourne la valeur de **MaVar** :

```
MonPointeur->
```

Dans ce cas, la chaîne "**Bonjour**" est retournée. Lorsque le symbole -> est placé derrière un pointeur, la valeur de l'objet vers lequel pointe le pointeur est récupérée. On dit alors qu'on **dépointe** le pointeur.

Il est important de comprendre que vous pouvez utiliser un pointeur suivi du symbole -> partout où vous auriez pu utiliser l'objet pointé lui-même. Vous pouvez placer l'expression **MonPointeur->** partout où vous pourriez utiliser la variable originale **MaVar**.

Par exemple, l'instruction suivante affiche une boîte de dialogue d'alerte comportant le mot Bonjour :

```
ALERTE(MonPointeur->)
```

Vous pouvez également utiliser **MonPointeur** pour modifier la valeur de **MaVar**. Par exemple, l'instruction suivante stocke la chaîne "**Au revoir**" dans la variable **MaVar** :

```
MonPointeur->:="Au revoir"
```

Si vous examinez les deux utilisations de l'expression **MonPointeur->** ci-dessus, vous constatez que cette expression se comporte exactement comme si vous aviez utilisé **MaVar** à sa place. En résumé : les deux lignes suivantes effectuent la même opération — elles affichent une boîte de dialogue d'alerte contenant la valeur courante de la variable **MaVar** :

```
ALERTE(MonPointeur->)
ALERTE(MaVar)
```

Les deux lignes suivantes effectuent la même opération ; elles assignent la chaîne "**Au revoir**" à **MaVar** :

```
MonPointeur->:="Au revoir"
MaVar:="Au revoir"
```

Utiliser des pointeurs vers des boutons

Ce paragraphe décrit l'utilisation d'un pointeur pour référencer un bouton. Un bouton (du point de vue du langage) n'est rien d'autre qu'une variable. Bien que les exemples de ce paragraphe utilisent des pointeurs pour référencer des boutons, les concepts présentés s'appliquent à l'utilisation de tout type d'objet

pouvant être référencé par un pointeur.

Imaginons que vous disposiez dans vos formulaires d'un certain nombre de boutons devant être actifs ou inactifs. Chaque bouton est associé à une condition qui peut être VRAI ou FAUX. La condition indique s'il faut désactiver ou activer le bouton. Vous pouvez utiliser un test comme celui-ci chaque fois que vous devez désactiver ou activer le bouton :

```
Si(Condition) ` Si la condition est VRAIE…
   OBJET FIXER ACTIVATION(MonBouton;Vrai) ` activer le bouton
Sinon ` Dans l'autre cas…
   OBJET FIXER ACTIVATION(MonBouton;Faux) ` désactiver le bouton
Fin de si
```

Vous pouvez avoir besoin, pour chaque bouton que vous avez créé, d'utiliser un test similaire, dans lequel seul le nom du bouton change. Afin d'être plus efficace, vous pouvez créer un pointeur pour référencer chaque bouton puis utiliser une sous-routine pour le test lui-même.

Vous devez utiliser des pointeurs si vous appelez une sous-routine, car il n'est pas possible de faire référence autrement aux variables des boutons. Par exemple, voici une méthode projet nommée ***FIXER_BOUTON***, qui référence un bouton avec un pointeur :

```
` Méthode projet FIXER_BOUTON
` FIXER_BOUTON ( Pointeur ; Booléen )
` FIXER_BOUTON ( -> Bouton ; Activé ou Désactivé )
`
` $1 - Pointeur vers un bouton
` $2 - Booléen. Si VRAI, active le bouton. Si FAUX, désactive le bouton

Si($2) ` Si la condition est VRAIE…
   OBJET FIXER ACTIVATION($1->;Vrai) ` activer le bouton
Sinon ` Si ce n'est pas le cas…
   OBJET FIXER ACTIVATION($1->;Faux) ` désactiver le bouton
Fin de si
```

Vous pouvez appeler la méthode projet ***FIXER_BOUTON*** de la manière suivante :

```
` ...
FIXER_BOUTON(->bValider;Vrai)
` ...
FIXER_BOUTON(->bValider;Faux)
` ...
FIXER_BOUTON(->bValider;([Employés]Nom#"")
` ...
Boucle($vlRadioBouton;1;20)
   $vpRadioBouton:=Pointeur vers("r"+Chaine($vlRadioBouton))
   FIXER_BOUTON($vpRadioBouton;Faux)
Fin de boucle
```

Utiliser des pointeurs vers des tables

Partout où le langage requiert un nom de table, vous pouvez utiliser un pointeur dépointé vers une table.
Pour créer un pointeur vers une table, écrivez une instruction du type :

```
TablePtr:=->[touteTable]
```

Vous pouvez également récupérer un pointeur vers une table à l'aide de la fonction **Table**. Par exemple :

```
TablePtr:=Table(20)
```

Vous pouvez utiliser le pointeur dépointé dans vos commandes, comme ceci :

```
TABLE PAR DEFAUT(TablePtr->)
```

Utiliser des pointeurs vers des champs

Partout où le langage requiert un nom de champ, vous pouvez utiliser un pointeur dépointé vers un champ.
Pour créer un pointeur vers un champ, écrivez une ligne d'instruction du type :

```
ChampPtr:=->[uneTable]CeChamp
```

Vous pouvez également récupérer un pointeur vers un champ à l'aide de la fonction **Champ**. Par exemple :

```
ChampPtr:=Champ(1;2)
```

Vous pouvez utiliser le pointeur dépointé avec les commandes, comme ceci :

```
OBJET FIXER POLICE(ChampPtr->;"Arial")
```

Utiliser des pointeurs vers des variables

L'exemple fourni au début de cette section illustre l'utilisation d'un pointeur vers une variable :

```
MaVar:="Bonjour" ` Création de la variable MaVar
MonPointeur:=->MaVar
```

Vous pouvez faire pointer des pointeurs vers des variables interprocess, process et, à compter de la version 2004.1 de 4D, vers des variables locales.

Lorsque vous utilisez des pointeurs vers des variables locales ou des variables process, vous devez veiller à ce que la variable pointée soit bien définie au moment de l'utilisation du pointeur. Rappelons que les variables locales sont supprimées à la fin de l'exécution de la méthode qui les a créées et les variables process à la fin du process dans lequel elles ont été créées. L'appel d'un pointeur vers une variable qui n'existe plus provoque une erreur de syntaxe en mode interprété (variable indéfinie) mais peut générer une erreur plus conséquente en mode compilé.

Note sur les variables locales : Les pointeurs vers des variables locales permettent dans de nombreux cas d'économiser des variables process. Les pointeurs vers des variables locales peuvent être utilisés uniquement à l'intérieur d'un même process.
Dans le débogueur, lorsque vous affichez un pointeur vers une variable locale déclarée dans une autre méthode, le nom de la méthode d'origine est indiquée entre parenthèses, derrière le pointeur. Par exemple, si vous écrivez dans Méthode1 :

```
$MaVar:="Bonjour"
Méthode2(->$MaVar)
```

Dans Méthode2, le débogueur affichera *$1* de la façon suivante :

$1 ->$MaVar (Méthode1)

La valeur de $1 sera :

$MaVar(Méthode1) "Bonjour"

Utiliser des pointeurs vers des éléments de tableau

Vous pouvez créer un pointeur vers un élément de tableau. Par exemple, les lignes d'instruction suivantes créent un tableau et assignent à une variable appelée **ElémPtr** un pointeur vers le premier élément :

```
TABLEAU REEL(unTableau;10) ` Créer un tableau
ElémPtr:=->unTableau{1} ` Créer un pointeur vers l'élément de tableau
```

Vous pouvez alors utiliser le pointeur dépointé pour assigner une valeur à l'élément, comme ceci :

```
ElémPtr->:=8
```

Utiliser des pointeurs vers des tableaux

Vous pouvez créer un pointeur vers un tableau. Par exemple, les lignes d'instruction suivantes créent un tableau et assignent à la variable nommée **TabPtr** un pointeur vers le tableau :

```
TABLEAU REEL(unTableau;10) ` Créer un tableau
TabPtr:=->unTableau ` Créer un pointeur vers le tableau
```

Il est important de comprendre que ce pointeur pointe vers le tableau, et non vers un élément du tableau. Par exemple, vous pourriez utiliser le pointeur dépointé de la manière suivante :

```
TRIER TABLEAU(TabPtr->;>) ` Tri du tableau
```

Si vous devez vous référer au quatrième élément du tableau à l'aide du pointeur, vous pouvez écrire :

```
TabPtr->{4}:=84
```

Utiliser un tableau de pointeurs

Il est souvent utile de disposer d'un tableau de pointeurs référençant un groupe d'objets homogène.

Un exemple d'utilisation typique d'un tel groupe d'objets est une grille de variables dans un formulaire. Chaque variable dans la grille est numérotée de manière séquentielle, par exemple : **Var1**, **Var2**,..., **Var10**. Vous devez souvent vous référer à ces variables, de manière indirecte — par leur numéro. Si vous créez un tableau de pointeurs et faites pointer les pointeurs vers chaque variable, il est alors facile de référencer les variables. Par exemple, pour créer un tableau et initialiser chaque élément, vous pouvez écrire :

```
TABLEAU POINTEUR(tabPointeurs;10) ` Créer un tableau de 10 pointeurs
Boucle($i;1;10) ` Boucle une fois par variable
   tabPointeurs{$i}:=Pointeur vers("Var"+Chaine($i))
 ` Initialiser chaque élément du tableau
Fin de boucle
```

La fonction **Pointeur vers** retourne un pointeur vers l'objet passé en paramètre.

Pour référencer une des variables, il suffit d'appeler les éléments du tableau. Par exemple, pour remplir les variables avec les dix prochains jours (en supposant que les variables soient de type Date), vous pourriez écrire :

```
Boucle($i;1;10) ` Boucle une fois par variable
   tabPointeurs{$i}->:=Date du jour+$i  ` Assigner les jours
Fin de boucle
```

Sélectionner un bouton avec un pointeur

Si vous disposez d'un groupe homogène de boutons radio dans un formulaire, vous devrez souvent pouvoir définir rapidement leur état. La solution consistant à référencer chacun d'entre eux par son nom n'est pas très pratique. Imaginons que vous disposiez d'un groupe de cinq boutons radio libellés **Bouton1**, **Bouton2**,..., **Bouton5**.

Dans un groupe de boutons radio, seul l'un d'entre eux est sélectionné. Le numéro du bouton sélectionné peut être stocké dans un champ numérique. Par exemple, si le champ *[Préférences]EtatBouton* contient 3, alors **Bouton3** est sélectionné. Dans la méthode formulaire, vous pouvez utiliser le code suivant pour paramétrer les boutons :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
   ` ...
      Au cas ou
         :([Préférences]EtatBouton=1)
            Bouton1:=1
         :([Préférences]EtatBouton=2)
            Bouton2:=1
         :([Préférences]EtatBouton=3)
            Bouton3:=1
         :([Préférences]EtatBouton=4)
            Bouton4:=1
         :([Préférences]EtatBouton=5)
            Bouton5:=1
      Fin de cas
   ` ...
Fin de cas
```

Un cas particulier doit être testé pour chaque bouton radio. La méthode peut être très longue si le formulaire contient de nombreux boutons radio. Heureusement, vous pouvez utiliser des pointeurs pour résoudre ce problème.
La fonction **Pointeur vers** vous permet de retourner un pointeur vers un bouton radio. L'exemple suivant utilise un pointeur de ce type pour référencer le bouton radio devant être sélectionné. Voici la méthode optimisée :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
   ` ...
      $vpRadio:=Pointeur vers("Bouton"+Chaine([Préférences]EtatBouton))
      $vpRadio->:=1
   ` ...
Fin de cas
```

Le numéro du bouton radio traité doit être stocké dans le champ *[Préférences]EtatBouton*. Cette opération peut être effectuée dans la méthode formulaire, pour l'événement formulaire Sur clic :

```
[Préférences]EtatBouton:=Bouton1+(Bouton2*2)+(Bouton3*3)+(Bouton4*4)+(Bouton5*5)
```

Passer des pointeurs aux méthodes

Vous pouvez passer un pointeur en tant que paramètre d'une méthode. A l'intérieur de la méthode, vous pouvez modifier l'objet référencé par le pointeur. Par exemple, la méthode suivante, **RECOIT DEUX**, reçoit deux paramètres qui sont des pointeurs. Elle passe l'objet référencé par le premier paramètre en caractères majuscules, et l'objet référencé par le second paramètre en caractères minuscules :

```
   ` Méthode projet RECOIT DEUX
   ` $1 - Pointeur vers un champ ou une variable de type Chaîne. Passe la chaîne en majuscules.
   ` $2 - Pointeur vers un champ ou une variable de type Chaîne. Passe la chaîne en minuscules.
$1->:=Majusc($1->)
$2->:=Minusc($2->)
```

L'instruction suivante emploie la méthode **RECOIT DEUX** pour passer un champ en caractères majuscules et une variable en caractères minuscules :

```
RECOIT DEUX(->[MaTable]MonChamp;->MaVar)
```

Si le champ, *[MaTable]MonChamp*, contenait la chaîne "*dupont*", celle-ci deviendrait "**DUPONT**". Si la variable **MaVar** contenait la chaîne "**BONJOUR**", celle-ci deviendrait "*bonjour*".

Dans la méthode **RECOIT DEUX** (et, en fait, à chaque fois que vous utilisez des pointeurs), il est important que les types de données des objets référencés soient corrects. Dans l'exemple précédent, les pointeurs doivent pointer vers des objets contenant une chaîne ou un texte.

Pointeurs vers des pointeurs

Si vous aimez compliquer les choses à l'extrême (bien que cela ne soit pas nécessaire dans 4D), vous pouvez utiliser des pointeurs pour référencer d'autres pointeurs. Examinons l'exemple suivant :

```
MaVar:="Bonjour"
PointeurUn:=->MaVar
PointeurDeux:=->PointeurUn
(PointeurDeux->)->:="Au revoir"
ALERTE((PointeurDeux->)->)
```

Cet exemple affiche une boîte de dialogue d'alerte contenant "**Au revoir**".
Voici la description de chaque ligne de l'exemple :

- *MaVar := "Bonjour"*
  --> Cette ligne place simplement la chaîne "**Bonjour**" dans la variable **MaVar**.

- *PointeurUn := ->MaVar*
  --> **PointeurUn** contient désormais un pointeur vers **MaVar**.

- *PointeurDeux :=->PointeurUn*
  --> **PointeurDeux** (une nouvelle variable) contient un pointeur vers **PointeurUn**, qui, elle, pointe vers **MaVar**.

- *(PointeurDeux->)-> := "Au revoir"*
  --> **PointeurDeux->** référence le contenu de **PointeurUn**, qui elle-même référence **MaVar**. Par conséquent, *(PointeurDeux->)->* référence le contenu de **MaVar**. Donc, dans ce cas, la valeur "**Au revoir**" est assignée à la **MaVar**.

- *ALERTE ((PointeurDeux->)->)*
  --> C'est ici la même chose que précédemment : **PointeurDeux->** référence le contenu de **PointeurUn**, qui elle-même référence **MaVar**. Par conséquent, *(PointeurDeux->)->* référence le contenu de **MaVar**. Donc, dans ce cas, la boîte de dialogue d'alerte affiche le contenu de *maVar*.

La ligne suivante place la valeur "**Bonjour**" dans **MaVar** :

```
(PointeurDeux->)->:="Bonjour"
```

La ligne suivante récupère "**Bonjour**" à partir de **MaVar** et la place dans **NouvelleVar** :

```
NouvelleVar:=(PointeurDeux->)->
```

Important : Vous devez utiliser des parenthèses lors des déréférencements multiples.

Nommer les objets du langage 4D

Cette section décrit les conventions d'écriture employées pour les nombreux objets du langage de 4D. Le nom de chaque objet doit respecter les règles suivantes :

- Un nom doit commencer par un caractère alphabétique (une lettre) ou un tiret bas.
- Le nom peut ensuite contenir des caractères alphabétiques, des caractères numériques, des espaces et des tirets bas (_).
- Les virgules, barres de fraction, guillemets et deux points (:) sont interdits.
- Les caractères réservés car utilisés comme opérateurs, comme l'astérisque (*) ou le +, sont interdits.
- 4D ignore les espaces superflus.

Note : Des règles supplémentaires sont à respecter lorsque les objets doivent être manipulés via le SQL : seuls les caractères *_0123456789abcdefghijklmnopqrstuvwxyz* sont acceptés, et le nom ne doit pas comporter de mot-clé SQL (commande, attribut, etc.). La zone "SQL" de l'inspecteur de l'éditeur de Structure signale automatiquement les caractères non autorisés dans un nom de table ou de champ.

Tables

Vous indiquez qu'un objet est une table en plaçant son nom entre crochets : [...]. Un nom de table peut contenir jusqu'à 31 caractères.

Exemples

```
TABLE PAR DEFAUT([Commandes])
FORM FIXER ENTREE([Clients];"Entrée")
AJOUTER ENREGISTREMENT([Lettres])
```

Champs

Vous indiquez qu'un objet est un champ en spécifiant d'abord la table à laquelle il appartient. Le nom du champ se place immédiatement derrière celui de la table. Un nom de champ peut contenir jusqu'à 31 caractères.

Exemples

```
[Commandes]Total:=Somme([Ligne]Montant)
CHERCHER([Clients];[Clients]Nom="Dupont")
[Lettres]Texte:=Capitaliser texte([Lettres]Texte)
```

Variables interprocess

Vous indiquez qu'un objet est une variable interprocess en faisant précéder son nom des symboles (<>), formés des caractères "inférieur à" suivi de "supérieur à".

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. De plus, sous Mac OS, vous pouvez également utiliser le caractère "diamant" (Option + v sur un clavier français).

Le nom d'une variable interprocess peut contenir jusqu'à 31 caractères, symbole <> non compris.

Exemples

```
<>vlProcessID:=Numero du process courant
<>vsKey:=Caractere(KeyCode)
Si(<>vtNom#"")
```

Variables process

Vous indiquez qu'un objet est une variable process en écrivant simplement son nom (qui ne doit pas commencer par les symboles $ ou <>). Ce nom peut contenir jusqu'à 31 caractères.

Exemples

```
vrGrandTotal:=Somme([Comptes]Montant)
Si(bValider=1)
   vsNomCourant:=""
```

Variables locales

Vous indiquez qu'un objet est une variable locale en faisant précéder son nom du symbole dollar ($). Le nom d'une variable locale peut contenir jusqu'à 31 caractères, signe dollar non compris.

Exemples

```
Boucle($vlEnregistrement;1;100)
   Si($vsTempVar="No")
      $vsMaChaîne:="Bonjour à tous"
```

Tableaux

Vous indiquez qu'un objet est un tableau en écrivant simplement son nom, qui est celui que vous passez à une commande de déclaration de tableau (par exemple **TABLEAU ENTIER LONG**) lorsque vous créez le tableau. Les tableaux sont des variables, et comme pour les variables, il existe trois types de tableaux qui se différencient par leur portée :

- Tableaux interprocess,
- Tableaux process,
- Tableaux locaux.

Tableaux interprocess

Le nom d'un tableau interprocess est précédé du symbole (<>), formé des caractères "inférieur à" suivi de "supérieur à".

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. De plus, sous Mac OS, vous pouvez également utiliser le caractère "diamant" (Option + v sur un clavier français).

Le nom d'un tableau interprocess peut contenir jusqu'à 31 caractères, symbole <> non compris.

Exemples

```
TABLEAU TEXTE(<>atSujets;Enregistrements dans table([Topics]))
TRIER TABLEAU(<>asMotsClés;>)
TABLEAU ENTIER(<>aiGrosTableau;10000)
```

Tableaux process

Vous indiquez qu'un objet est un tableau process en écrivant simplement son nom (qui ne doit pas commencer par les symboles $ ou <>). Ce nom peut contenir jusqu'à 31 caractères.

Exemples

```
TABLEAU TEXTE(atSujets;Enregistrements dans table([Topics]))
TRIER TABLEAU(asMotsClés;>)
TABLEAU ENTIER(aiGrosTableau;10000)
```

Tableaux locaux

Un tableau est déclaré local lorsque son nom est précédé du signe dollar ($). Le nom d'un tableau local peut contenir jusqu'à 31 caractères, signe dollar non compris.

Exemples

```
TABLEAU TEXTE($atSujets;Enregistrements dans table([Topics]))
TRIER TABLEAU($asMotsClés;>)
TABLEAU ENTIER($aiGrosTableau;10000)
```

Eléments de tableaux

Vous désignez un élément d'un tableau local, process ou interprocess à l'aide d'accolades ({…}). L'élément référencé (l'indice) est indiqué par une expression numérique.

Exemples

```
  ` Adresser un élément d'un tableau interprocess
Si(<>asMotsClés{1}="Stop")
   <>atSujets{$vlElem}:=[Topics]Sujet
   $viValeurSuivante:=<>aiGrosTableau{Taille tableau(<>aiGrosTableau)}

  ` Adresser un élément d'un tableau process
   Si(asMotsClés{1}="Stop")
      atSujets{$vlElem}:=[Topics]Sujet
      $viValeurSuivante:=aiGrosTableau{Taille tableau(aiGrosTableau)}

  ` Adresser un élément d'un tableau local
      Si($asMotsClés{1}="Stop")
         $atSujets{$vlElem}:=[Topics]Sujet
         $viValeurSuivante:=$aiGrosTableau{Taille tableau($aiGrosTableau)}
```

Eléments de tableaux à deux dimensions

Vous désignez un élément d'un tableau à deux dimensions à l'aide d'une double paire d'accolades ({…}). L'élément référencé (l'indice) est indiqué par deux expressions numériques dans deux paires d'accolades.

Exemples

```
  ` Adresser un élément d'un tableau interprocess à deux dimensions
Si(<>asMotsClés{$vlLigneSuivante}{1}="Stop")
   <>atSujets{10}{$vlElem}:=[Topics]Sujet
   $viValeurSuivante:=<>aiGrosTableau{$vlSet}{Taille tableau(<>aiGrosTableau{$vlSet})}

  ` Adresser un élément d'un tableau process à deux dimensions
   Si(asMotsClés{$vlLigneSuivante}{1}="Stop")
      atSujets{10}{$vlElem}:=[Topics]Sujet
      $viValeurSuivante:=aiGrosTableau{$vlSet}{Taille tableau(aiGrosTableau{$vlSet})}

  ` Adresser un élément d'un tableau interprocess à deux dimensions
      Si($asMotsClés{$vlLigneSuivante}{1}="Stop")
         $atSujets{10}{$vlElem}:=[Topics]Sujet
         $viValeurSuivante:=$aiGrosTableau{$vlSet}{Taille tableau($aiGrosTableau{$vlSet})}
```

Formulaires

Vous indiquez qu'un objet est un formulaire en utilisant une expression de type chaîne alphanumérique qui représente son nom. Le nom d'un formulaire peut contenir jusqu'à 31 caractères.

Exemples

```
FORM FIXER ENTREE([Personnes];"Entrée")
FORM FIXER SORTIE([Personnes];"Sortie")
DIALOGUE([Stock];"Boîte de note"+Chaine($vlStage))
```

Objets de formulaires

Vous désignez un objet de formulaire en passant son nom sous forme de chaîne, précédée du paramètre *. Un nom d'objet peut contenir jusqu'à 255 octets.

Exemple :

```
OBJET FIXER POLICE(*;"Binfo";"Times")
```

Voir aussi la section **Objets de formulaires**.

Méthodes

Vous indiquez qu'un objet est une méthode (sous-routine ou fonction utilisateur) en saisissant son nom. Ce nom peut contenir jusqu'à 31 caractères.

Note : Une méthode qui ne retourne pas de résultat est appelée une procédure. Une méthode qui retourne un résultat est appelée une fonction utilisateur.

Exemples

```
Si(Nouveau client)
   EFFACER VALEURS DUPLIQUEES
   APPLIQUER A SELECTION([Employés];AUGMENTER SALARIES)
```

Conseil : Nous vous recommandons d'adopter, pour nommer vos méthodes, la même convention que celle utilisée dans le langage de 4D : écrivez les noms de vos procédures en caractères majuscules, et vos fonctions en minuscules avec la première lettre en majuscule. Ainsi, lorsque vous réouvrirez une base au bout de plusieurs mois, vous identifierez immédiatement si une méthode retourne ou non un résultat, en regardant son nom dans la fenêtre de l'Explorateur.

Note : Lorsque vous souhaitez appeler une méthode, vous saisissez simplement son nom. Toutefois, certaines commandes intégrées telles que **APPELER SUR EVENEMENT**, ainsi que les commandes des plug-ins, nécessitent que vous passiez le nom d'une méthode en tant que chaîne lorsqu'un paramètre de type méthode est requis (cf. exemples ci-dessous).

Exemples

```
  ` Cette commande attend une méthode (fonction) ou une formule
CHERCHER PAR FORMULE([aTable];Recherche Spéciale)
  ` Cette commande attend une méthode (procédure) ou une formule
APPLIQUER A SELECTION([Employés];AUGMENTER SALARIES)
  ` Mais cette commande attend un nom de méthode
APPELER SUR EVENEMENT("GERER EVENEMENTS")
  ` Et cette commande de plug-in attend un nom de méthode
WR APPELER SUR ERREUR("WR GERER ERREURS")
```

Les méthodes peuvent accepter des paramètres (ou arguments). Le paramètres sont passés à la méthode entre parenthèses, à la suite du nom de la méthode. Les paramètres sont séparés par des points virgule (;). Les paramètres sont passés à la méthode appelée en tant que variables locales numérotées séquentiellement : *$1, $2,..., $n*. De plus, plusieurs paramètres consécutifs (s'ils sont les derniers) peuvent être adressés à l'aide de la syntaxe *${n}*où n, expression numérique, est le numéro du paramètre.
A l'intérieur d'une fonction, la variable locale *$0* contient la valeur à retourner.

Exemples

```
  ` Dans ELIMINER ESPACES, $1 est pointeur sur le champ [Personnes]Nom
ELIMINER ESPACES(->[Personnes]Nom)

  ` Dans Créateur tableau :
  ` - $1 est un numérique qui vaut 1
  ` - $2 est un numérique qui vaut 5
  ` - $3 est un texte ou un alpha qui vaut "Super"
  ` - La valeur résultante est assignée à $0
$vsRésultat:=Créateur tableau(1;5;"Super")

  ` Dans Poubelle :
  ` - Les trois paramètres sont de type Texte ou Alpha
  ` - Vous pouvez y accéder par $1, $2 ou $3
  ` - Vous pouvez y accéder en écrivant, par exemple, ${$vlParam} où $vlParam vaut 1, 2 ou 3
  ` - La valeur résultante est assignée à $0
vtClone:=Poubelle("est";"le";"il")
```

Commandes de plug-ins (procédures, fonctions et zones externes)

Vous indiquez qu'un objet est une commande de plug-in en écrivant son nom tel qu'il est défini dans le plug-in. Le nom d'une commande de plug-in peut

contenir jusqu'à 31 caractères.

Exemples

```
WR SUPPRIMER SELECTION(wrZone)
$pvNouvelleZone:=PV Creer zone hors ecran
```

Ensembles

Dans 4D, il existe deux types d'ensembles qui se distinguent par leur portée :

- Ensembles interprocess,
- Ensembles process.

On peut également distinguer un troisième type d'ensemble, spécifique à 4D Server:

- Ensembles clients.

Ensembles interprocess

Un ensemble est déclaré interprocess lorsque son nom, qui est une expression de type chaîne alphanumérique, est précédé du symbole (<>), formé des caractères “inférieur à” suivi de “supérieur à”.

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. De plus, sous Mac OS, vous pouvez également utiliser le caractère "diamant" (Option + v sur un clavier français).

Le nom d’un ensemble interprocess peut comporter jusqu’à 255 caractères, symbole <> non compris.

Ensembles process

Vous déclarez un ensemble process en passant simplement une expression de type chaîne qui représente son nom (et qui ne doit pas débuter par les symboles <> ou $). Le nom d’un ensemble process peut comporter jusqu’à 255 caractères.

Ensembles client

Le nom d'un ensemble client doit être précédé du symbole dollar ($). Ce nom peut comporter jusqu'à 255 caractères, symbole dollar non compris.

Note : Les ensembles sont gérés par le serveur. Dans certains cas, pour des raisons particulières ou d'optimisation, vous pourrez avoir besoin d'utiliser des ensembles localement, sur les postes clients. Pour cela, il vous suffit de créer des ensembles client.

Exemples

```
  ` Ensembles interprocess
UTILISER ENSEMBLE("<>Enregistrements supprimés")
NOMMER ENSEMBLE([Clients];"<>Commandes clients")
Si(Enregistrements dans ensemble("<>Sélection"+Chaine($i))>0)
  ` Ensembles process
   UTILISER ENSEMBLE("Enregistrements supprimés")
   NOMMER ENSEMBLE([Clients];"Commandes clients")
   Si(Enregistrements dans ensemble("Sélection"+Chaine($i))>0)
  ` Ensembles client
      UTILISER ENSEMBLE("$Enregistrements supprimés")
      NOMMER ENSEMBLE([Clients];"$Commandes clients")
      Si(Enregistrements dans ensemble("$Sélection"+Chaine($i))>0)
```

Sélections temporaires

Dans 4D, il existe deux types de sélections temporaires, qui se distinguent par leur portée :

- Sélections temporaires interprocess,
- Sélections temporaires process.

Sélections temporaires interprocess

Vous désignez une sélection temporaire interprocess en utilisant une expression de type chaîne débutant par le symbole (<>), formé des caractères “inférieur à” suivi de “supérieur à”.

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. De plus, sous Mac OS, vous pouvez également utiliser le caractère "diamant" (Option + v sur un clavier français).

Le nom d'une sélection temporaire interprocess peut contenir jusqu’à 255 caractères, symbole <> non compris.

Sélections temporaires process

Vous déclarez une sélection temporaire process en passant simplement une expression de type chaîne qui représente son nom (et qui ne doit pas débuter par les symboles <> ou $). Le nom d’une sélection temporaire process peut comporter jusqu’à 255 caractères.

Exemples

```
  ` Sélection temporaire interprocess
UTILISER SELECTION([Clients];"<>ParCodePostal")
  ` Sélection temporaire process
UTILISER SELECTION([Clients];"ParCodePostal")
```

Process

En mode mono-utilisateur, ou sur le poste client en mode client/serveur, il existe deux types de process :

- Process globaux,
- Process locaux.

Process globaux

Vous déclarez un process global en passant une expression de type chaîne de caractères qui représente son nom (qui ne doit pas commencer par le symbole *$*). Le nom d'un process peut comporter jusqu'à 255 caractères.

Process locaux

Vous déclarez un process local lorsque son nom est précédé du symbole dollar (*$*). Le nom d'un process peut comporter jusqu'à 255 caractères, symbole dollar non compris.

Exemple

```
  ` Lancer le process global "Ajouter Clients"
$vlProcessID:=Nouveau process("P_AJOUT_CLIENTS";48*1024;"Ajouter Clients")
  ` Lancer le process local "$Suivre Souris"
$vlProcessID:=Nouveau process("P_SUIVRE_SOURIS";16*1024;"$Suivre Souris")
```

Résumé des conventions d'écriture dans 4D

Le tableau suivant résume les conventions utilisées par 4D pour nommer les objets dans les méthodes.

Objet	Longueur max.	Exemple
Table	31	[Factures]
Champ	31	[Employés]Nom
Variable interprocess	<> + 31	<>vlProcessSuivantID
Variable process	31	vsNomCourant
Variable locale	$ + 31	$vlCompteurLocal
Formulaire	31	"Formulaire Web perso"
Objet de formulaire	31	"MonBouton"
Tableau interprocess	<> + 31	<>apTableaux
Tableau process	31	asGenre
Tableau local	$ + 31	$atValeurs
Méthode	31	M_AJOUTER_CLIENTS
Commande de plug-in	31	WR INSERER TEXTE
Ensemble interprocess	<> + 255	"<>Enregistrements à archiver"
Ensemble process	255	"Enregistrements actuels"
Ensemble client	$ + 255	"$Sujets précédents"
Sélection temporaire	255	"Employés de A à Z"
Sélection temporaire interprocess	<> + 255	"<>Employés de Z à A"
Process local	$ + 255	"$SuivreEvénements"
Process global	255	"P_MODULE_FACTURES"
Sémaphore	255	"monsémaphore"

Résoudre les conflits de noms

Si un objet d'un certain type a le même nom qu'un autre objet d'un autre type (par exemple, si un champ est baptisé Personnes et qu'une variable est également nommée Personnes), 4D utilise un système de priorités pour identifier l'objet. Veillez à utiliser des noms uniques pour les différents éléments de votre base.

4D identifie les noms utilisés dans les méthodes en fonction de l'ordre de priorité suivant :

1. Champs
2. Commandes
3. Méthodes
4. Routines de plug-ins
5. Constantes prédéfinies
6. Variables

Par exemple, 4D dispose d'une fonction interne appelée **Date**. Si vous appelez **Date** une de vos méthodes, 4D considérera "Date" comme étant la fonction interne et non votre méthode. Vous ne pourrez pas appeler votre méthode. En revanche, si vous nommez un champ "Date", 4D considérera que vous souhaitez appeler votre champ et non la fonction intégrée.

Quelle que soit la simplicité ou la complexité d'une méthode, vous utiliserez toujours un ou plusieurs types de structure de programmation. Les structures de programmation déterminent si et dans quel ordre les lignes d'instructions sont exécutées à l'intérieur d'une méthode. Il existe trois types de structures :

- séquentielle,
- conditionnelle,
- répétitive.

Le langage de 4D dispose d'instructions permettant de contrôler chacune de ces structures.

Structures séquentielles

Une structure séquentielle est une structure simple, linéaire. Une séquence est une série d'instructions que 4D exécute les unes après les autres, de la première à la dernière. Par exemple :

```
FORMULAIRE SORTIE([Personnes];"Listing")
TOUT SELECTIONNER([Personnes])
VISUALISER SELECTION([Personnes])
```

Une instruction d'une ligne, fréquemment utilisée pour les méthodes objet, est le cas le plus simple de structure séquentielle. Par exemple :

```
[Personnes]Nom:=Majusc([Personnes]Nom)
```

Note : Les mots-clés **Debut SQL** / **Fin SQL** permettent de délimiter des structures séquentielles destinées à être exécutées par le moteur SQL de 4D. Pour plus d'informations, reportez-vous à la description de ces mots-clés.

Structures conditionnelles

Une structure conditionnelle permet aux méthodes de tester une condition et d'exécuter des séquences d'instructions différentes en fonction du résultat. La condition est une expression booléenne, c'est-à-dire pouvant retourner VRAI ou FAUX. L'une des structures conditionnelles est la structure **Si...Sinon...Fin de si**, qui aiguille le déroulement du programme vers une séquence ou une autre. L'autre structure conditionnelle est la structure **Au cas ou...Sinon...Fin de cas**, qui aiguille le programme vers une séquence parmi une ou plusieurs alternatives.

Structures répétitives (ou "boucles")

Il est très courant, lorsque vous écrivez des méthodes, de rencontrer des cas où vous devez répéter une séquence d'instructions un certain nombre de fois. Pour traiter ces besoins, le langage vous propose trois structures répétitives : **Tant que...Fin tant que**, **Repeter...Jusque**, et **Boucle...Fin de boucle**. Les boucles sont contrôlées de deux manières : soit elles se répètent jusqu'à ce qu'une condition soit remplie, soit elles se répètent un nombre fixé de fois. Chaque structure répétitive peut être utilisée de l'une ou l'autre manière, mais les boucles Tant que et Repeter sont mieux adaptées à la répétition jusqu'à ce qu'une condition soit remplie, alors que les boucles Boucle sont mieux adaptées à la répétition un certain nombre de fois.

Note : 4D vous permet d'imbriquer des structures de programmation (Si/Tant que/Boucle/Au cas ou/Repeter) jusqu'à une "profondeur" de 512 niveaux.

Si...Sinon...Fin de si

La syntaxe de la structure conditionnelle **Si...Sinon...Fin de si** est la suivante :

```
Si(Expression_booléenne)
   instruction(s)
Sinon
   instruction(s)
Fin de si
```

A noter que l'élément **Sinon** est optionnel, vous pouvez écrire :

```
Si(Expression_booléenne)
   instruction(s)
Fin de si
```

La structure **Si...Sinon...Fin de si** permet à votre méthode de choisir dans une alternative, en fonction du résultat, VRAI ou FAUX, d'un test (une expression booléenne).

Si l'expression booléenne est VRAIE, les instructions qui suivent immédiatement le test sont exécutées. Si l'expression booléenne est FAUSSE, les instructions suivant la ligne **Sinon** sont exécutées. Le **Sinon** est optionnel ; lorsqu'il est omis, c'est la première ligne d'instructions suivant le **Fin de si** (s'il y en a une) qui est exécutée.

A noter que l'expression booléenne est toujours évaluée en totalité. Examinons en particulier le test suivant :

```
Si(MéthodeA & MéthodeB)
   ...
Fin de si
```

L'expression n'est VRAIE que si les deux méthodes sont VRAIES. Or, même si ***MéthodeA*** retourne FAUX, 4D évaluera quand même ***MéthodeB***, ce qui représente une perte de temps inutile. Dans ce cas, il est préférable d'utiliser une structure du type :

```
Si(MéthodeA)
   Si(MéthodeB)
      ...
   Fin de si
Fin de si
```

Le résultat est équivalent et ***MéthodeB*** n'est évaluée que si nécessaire.

Exemple

```
  ` Demander à l'utilisateur de saisir un nom
$Rech:=Demander(Saisissez un nom)
Si(OK=1)
   CHERCHER([Personnes];[Personnes]Nom=$Rech)
Sinon
   ALERTE("Vous n'avez pas saisi de nom.")
Fin de si
```

Astuce : Il n'est pas obligatoire que des instructions soient exécutées dans chaque branche de l'alternative. Lorsque vous développez un algorithme, ou lorsque vous poursuivez un but précis, rien ne vous empêche d'écrire :

```
Si(Expression_booléenne)
Sinon
   instruction(s)
Fin de si
```

ou :

```
Si(Expression_booléenne)
   instruction(s)
Sinon
Fin de si
```

Au cas ou...Sinon...Fin de cas

La syntaxe de la structure conditionnelle **Au cas ou...Sinon...Fin de cas** est la suivante :

```
Au cas ou
   :(Expression_booléenne)
      instruction(s)
   :(Expression_booléenne)
      instruction(s)
      .
      .
      .

   :(Expression_booléenne)
      instruction(s)
   Sinon
      instruction(s)
Fin de cas
```

A noter que l'élément **Sinon** est optionnel, vous pouvez donc écrire :

```
Au cas ou
   :(Expression_booléenne)
      instruction(s)
   :(Expression_booléenne)
      instruction(s)
      .
      .
      .

   :(Expression_booléenne)
      instruction(s)
Fin de cas
```

Tout comme la structure **Si...Sinon...Fin de si**, la structure **Au cas ou...Sinon...Fin de cas** permet également à votre méthode de choisir parmi plusieurs séquences d'instructions. A la différence de la structure précédente, le **Au cas ou...Sinon...Fin de cas** peut tester un nombre illimité d'expressions booléennes et exécuter la séquence d'instructions correspondant à la valeur VRAI.

Chaque expression booléenne débute par le caractère deux points (:). La combinaison de deux points et d'une expression booléenne est appelé un cas. Par exemple, la ligne suivante est un cas :

```
:(bValider=1)
```

Seules les instructions suivant le premier cas VRAI (et ce, jusqu'au cas suivant) seront exécutées. Si aucun des cas n'est VRAI, aucune instruction n'est exécutée (s'il n'y a pas d'élément **Sinon**).

Vous pouvez placer une instruction **Sinon** après le dernier cas. Si tous les cas sont FAUX, les instructions suivant le **Sinon** seront exécutées.

Exemple

Cet exemple teste une variable numérique et affiche une boîte de dialogue d'alerte comportant un simple mot :

```
Au cas ou
   :(vRésult=1) ` Test si le numéro est 1
      ALERTE("Un.") ` Si c'est 1, afficher une alerte
   :(vRésult=2) ` Test si le numéro est 2
      ALERTE("Deux.") ` Si c'est 2, afficher une alerte
   :(vRésult=3) ` Test si le numéro est 3
      ALERTE("Trois.") ` Si c'est 3, afficher une alerte
   Sinon ` Si ce n'est ni 1 ni 2 ni 3, afficher une alerte
      ALERTE("Ce n'est ni un, ni deux, ni trois.")
Fin de cas
```

A titre de comparaison, voici la version avec **Si...Sinon...Fin de si** de la même méthode :

```
Si(vRésult=1) ` Test si le numéro est 1
   ALERTE("Un.") ` Si c'est 1, afficher une alerte
Sinon
   Si(vRésult=2) ` Test si le numéro est 2
      ALERTE("Deux.") ` Si c'est 2, afficher une alerte
   Sinon
      Si(vRésult=3) ` Test si le numéro est 3
         ALERTE("Trois.") ` Si c'est 3, afficher une alerte
      Sinon ` Si ce n'est ni 1, 2 ni 3, afficher l'alerte
         ALERTE("Ce n'est ni un, ni deux, ni trois.")
```

```
      Fin de si
   Fin de si
Fin de si
```

Rappelez-vous qu'avec une structure de type **Au cas ou...Sinon...Fin de cas**, seul le premier cas VRAI rencontré est exécuté. Même si d'autres cas sont VRAIS, seules les instructions suivant le premier cas VRAI seront prises en compte.

Par conséquent, lorsque vous testez dans la même méthode des cas simples et des cas complexes, vous devez placer les cas complexes **avant** les cas simples, sinon ils ne seront jamais exécutés. Par exemple, si vous souhaitez traiter le cas simple *(vRésult=1)* et le cas complexe *(vRésult=1) & (vDemande#2)* et que vous structurez la méthode de la manière suivante :

```
Au cas ou
   :(vRésult=1)
      ... `instruction(s)
   :((vRésult=1) & (vDemande#2)) `<-- Les instructions suivant ce cas ne seront JAMAIS exécutées
      ... `instruction(s)
Fin de cas
```

... les instructions associées au cas complexe ne seront jamais exécutées. En effet, pour que ce cas soit VRAI, ses deux conditions booléennes doivent l'être. Or, la première condition est celle du cas simple situé précédemment. Lorsqu'elle est VRAIE, le cas simple est exécuté et 4D sort de la structure conditionnelle, sans évaluer le cas complexe. Pour que ce type de méthode fonctionne, vous devez écrire :

```
Au cas ou
   :(vRésult=1) & (vDemande#2) `<-- Les cas complexes doivent toujours être placés en premier
      ... `Instruction(s)
   :(vRésult=1)
      ... `Instruction(s)
Fin de cas
```

Astuce : Il n'est pas obligatoire que des instructions soient exécutées dans toutes les alternatives. Lorsque vous développez un algorithme, ou lorsque vous poursuivez un but particulier, rien ne vous empêche d'écrire :

```
Au cas ou
   :(Expression_booléenne)
   :(Expression_booléenne)

      .
      .
      .

   :(Expression_booléenne)
      instruction(s)
   Sinon
      instruction(s)
Fin de cas
```

ou :

```
Au cas ou
   :(Expression_booléenne)
   :(Expression_booléenne)
      instruction(s)
      .
      .
      .

   :(Expression_booléenne)
      instruction(s)
   Sinon
Fin de cas
```

ou :

```
Au cas ou
   Sinon
      instruction(s)
Fin de cas
```

Tant que...Fin tant que

La syntaxe de la structure répétitive (ou boucle) **Tant que...Fin tant que** est la suivante :

```
Tant que(Expression_booléenne)
   instruction(s)
Fin tant que
```

Une boucle **Tant que...Fin tant que** exécute les instructions comprises entre **Tant que** et **Fin tant que** aussi longtemps que l'expression booléenne est VRAIE. Elle teste l'expression booléenne initiale et n'entre pas dans la boucle (et donc n'exécute aucune instruction) si l'expression est à FAUX.

Il est utile d'initialiser la valeur testée dans l'expression booléenne juste avant d'entrer dans la boucle Tant que. Initialiser la valeur signifie lui affecter un contenu approprié, généralement pour que l'expression booléenne soit VRAIE et que le programme entre dans la boucle.

La valeur de l'expression booléenne doit pouvoir être modifiée par un élément situé à l'intérieur de la boucle, sinon elle s'exécutera indéfiniment. La boucle suivante est sans fin car Infini est toujours VRAI :

```
Infini:=Vrai
Tant que(Infini)
Fin tant que
```

Si vous vous retrouvez dans une telle situation (où une méthode s'exécute de manière incontrôlée), vous pouvez utiliser les fonctions de débogage de 4D et remonter à la source du problème. Pour plus d'informations sur ce point, reportez-vous à la section **Débogueur**.

Exemple

```
CONFIRMER("Ajouter un enregistrement?") ` Est-ce que l'utilisateur veut ajouter un enregistrement?
Tant que(OK=1) ` Tant que l'utilisateur accepte
   AJOUTER ENREGISTREMENT([Table]) ` Ajouter un nouvel enregistrement
Fin tant que ` Une boucle Tant que se termine toujours par Fin tant que
```

Dans cet exemple, la valeur de la variable système OK est définie par la commande **CONFIRMER** avant que le programme n'entre dans la boucle. Si l'utilisateur clique sur le bouton OK dans la boîte de dialogue de confirmation, la variable OK prend la valeur 1 et la boucle est exécutée. Dans le cas contraire, la variable OK prend la valeur 0 et la boucle est ignorée. Une fois que le programme entre dans la boucle, la commande **AJOUTER ENREGISTREMENT** permet de continuer à l'exécuter car elle met la variable système OK à 1 lorsque l'utilisateur sauvegarde l'enregistrement. Lorsque l'utilisateur annule (ne valide pas) le dernier enregistrement, la variable système OK prend la valeur 0 et la boucle s'arrête.

Repeter...Jusque

La syntaxe de la structure répétitive (ou boucle) **Repeter...Jusque** est la suivante :

```
Repeter
   instruction(s)
Jusque(Expression_booléenne)
```

La boucle **Repeter...Jusque** est semblable à la boucle **Tant que...Fin tant que**, à la différence qu'elle teste la valeur de l'expression booléenne après l'exécution de la boucle et non avant. Ainsi, la boucle est toujours exécutée au moins une fois, tandis que si l'expression booléenne est initialement à FAUX, la boucle **Tant que...Fin tant que** ne s'exécute pas du tout.

L'autre particularité de la boucle **Repeter...Jusque** est qu'elle se poursuit jusqu'à ce que l'expression booléenne soit à VRAI.

Exemple

Comparez l'exemple suivant avec celui de la boucle **Tant que...Fin tant que** : vous constatez qu'il n'est pas nécessaire d'initialiser l'expression booléenne — il n'y a pas de commande **CONFIRMER** pour initialiser la variable OK.

```
Repeter
   AJOUTER ENREGISTREMENT([aTable])
Jusque(OK=0)
```

Boucle...Fin de boucle

La syntaxe de la structure répétitive **Boucle...Fin de boucle** est la suivante :

```
Boucle(Variable_Compteur;Expression_Début;Expression_Fin{;Expression_Incrément})
   instructions(s)
Fin de boucle
```

La structure **Boucle...Fin de boucle** est une boucle contrôlée par un compteur :

- La variable compteur **Variable_Compteur** est une variable numérique (Réel, Entier ou Entier long) initialisée par **Boucle...Fin de boucle** à la valeur spécifiée par **Expression_Début**.
- La variable **Variable_Compteur** est incrémentée de la valeur spécifiée par le paramètre optionnel **Expression_Incrément** à chaque fois que la boucle est exécutée. Si vous ne passez pas de valeur dans **Expression_Incrément**, la variable compteur est incrémentée par défaut de un (*1*).
- Lorsque le compteur atteint la valeur définie par **Expression_Fin**, la boucle s'arrête.

Important : Les expressions numériques **Expression_Début**, **Expression_Fin** et **Expression_Incrément** sont évaluées une seule fois, au début de la boucle. Si ces expressions sont des variables, leur modification depuis l'intérieur de la boucle **n'affectera pas l'exécution de la boucle**.

Astuce : En revanche, vous pouvez, si vous le souhaitez, modifier la valeur de la variable **Variable_Compteur** depuis l'intérieur de la boucle et cela **affectera l'exécution de la boucle**.

- Généralement, **Expression_Début** est inférieure à **Expression_Fin**.
- Si les deux expressions sont égales, la boucle ne sera exécutée qu'une fois.
- Si **Expression_Début** est supérieure à **Expression_Fin**, la boucle ne s'exécutera pas du tout, à moins que vous ne spécifiiez une **Expression_Incrément** négative. Reportez-vous ci-dessous au paragraphe décrivant ce point.

Exemples élémentaires

(1) La boucle suivante s'exécute 100 fois :

```
Boucle(vCompteur;1;100)
  ` Faire quelque chose
Fin de boucle
```

(2) L'exemple suivant permet de traiter tous les éléments du tableau *unTableau* :

```
Boucle($vlElem;1;Taille tableau(unTableau))
  ` Faire quelque chose avec l'élément
   unTableau{$vlElem}:=...
Fin de boucle
```

(3) L'exemple suivant permet d'examiner chaque caractère du texte *vtDuTexte* :

```
Boucle($vlCar;1;Longueur(vtDuTexte))
  ` Faire quelque chose avec le caractère si c'est une tabulation
   Si(Code de caractere(vtDuTexte≤$vlCar≥)=Tabulation
  ` ...
   Fin de si
Fin de boucle
```

(4) L'exemple suivant permet de traiter tous les enregistrements de la sélection de la table *[uneTable]*:

```
DEBUT SELECTION([uneTable])
Boucle($vlEnrg;1;Enregistrements trouves([uneTable]))
  ` Faire quelque chose avec chaque enregistrement
   ENVOYER ENREGISTREMENT([uneTable])
  ` ...
  ` Passer à l'enregistrement suivant
   ENREGISTREMENT SUIVANT([uneTable])
Fin de boucle
```

La plupart des structures **Boucle...Fin de boucle** que vous écrirez dans vos bases ressembleront à celles présentées ci-dessus.

Décrémenter la variable Compteur

Dans certains cas, vous pouvez souhaiter disposer d'une boucle dont la valeur de la variable compteur décroît au lieu de croître. Pour cela, **Expression_Début** doit être supérieure à **Expression_Fin** et **Expression_Increment** doit être négative. Les exemples suivants effectuent les mêmes tâches que les précédents, mais en sens inverse :

(5) La boucle suivante s'exécute 100 fois :

```
Boucle(vCompteur;100;1;-1)
  ` Faire quelque chose
Fin de boucle
```

(6) L'exemple suivant permet de traiter tous les éléments du tableau *unTableau* :

```
Boucle($vlElem;Taille tableau(unTableau);1;-1)
  ` Faire quelque chose avec l'élément
   unTableau{$vlElem}:=...
Fin de boucle
```

(7) L'exemple suivant permet d'examiner chaque caractère du texte *vtDuTexte* :

```
Boucle($vlCar;Longueur(vtDuTexte);1;-1)
  ` Faire quelque chose avec le caractère si c'est une tabulation
   Si(Code de caractere(vtDuTexte≤$vlCar≥)=Tabulation)
  ` ...
   Fin de si
Fin de boucle
```

(8) L'exemple suivant permet de traiter tous les enregistrements de la sélection de la table *[uneTable]*:

```
ALLER A DERNIER ENREGISTREMENT([uneTable])
Boucle($vlEnrg;Enregistrements trouves([uneTable]);1;-1)
  ` Faire quelque chose avec chaque enregistrement
   ENVOYER ENREGISTREMENT([uneTable])
  ` ...
  ` Passer à l'enregistrement précédent
   ENREGISTREMENT PRECEDENT([uneTable])
Fin de boucle
```

Incrementer la variable compteur de plus de 1

Si vous le souhaitez, vous pouvez passer dans **Expression_Incrément** une valeur (positive ou négative) dont la valeur absolue est supérieure à un.

(9) La boucle suivante ne traite que les éléments pairs du tableau *unTableau* :

```
Boucle($vlElem;2;Taille tableau(unTableau);2)
  ` Faire quelque chose avec l'élément 2,4...2n
   unTableau{$vlElem}:=...
Fin de boucle
```

Sortir d'une boucle en modifiant la variable compteur

Dans certains cas, vous voudrez exécuter une boucle un certain nombre de fois, mais également pouvoir sortir si une autre condition devient Vraie.

Pour cela, il vous suffit de tester cette condition à l'intérieur de la boucle et, si elle devient Vraie, de "forcer" la valeur de la variable compteur, de manière à ce qu'elle soit supérieure à celle de **Expression_Fin**.

(10) Dans l'exemple suivant, vous effectuez une boucle parmi les enregistrements d'une sélection jusqu'à ce que la fin de la sélection soit atteinte, ou bien jusqu'à ce que la variable interprocess *<>vbStop*, initialement fixée à Faux, prenne la valeur Vrai. Cette variable est gérée par une méthode projet ***APPELER SUR EVENEMENT*** utilisée pour interrompre l'opération :

```
<>vbStop:=Faux
APPELER SUR EVENEMENT("GESTION STOP")
  ` GESTION STOP définit <>vbStop à Vrai si les touches Ctrl+point (Windows) ou Commande+point (Mac OS) sont
enfoncées
$vlNbEnrgs:=Enregistrements trouves([aTable])
DEBUT SELECTION([aTable])
Boucle($vlEnrgs;1;$vlNbEnrgs)
  ` Faire quelque chose avec l'enregistrement
   ENVOYER ENREGISTREMENT([aTable])
  ` ...
  ` Aller à l'enregistrement suivant
   Si(<>vbStop)
      $vlEnrgs:=$vlNbEnrgs+1 ` Forcer la variable compteur à stopper la boucle
   Sinon
      ENREGISTREMENT SUIVANT([aTable])
   Fin de si
Fin de boucle
APPELER SUR EVENEMENT("")
Si(<>vbStop)
   ALERTE("L'opération a été interrompue.")
Sinon
   ALERTE("L'opération s'est terminée avec succès.")
Fin de si
```

Comparaison des structures répétitives

Reprenons le premier exemple employé pour la structure **Boucle...Fin de boucle** :

(1) La boucle suivante s'exécute 100 fois :

```
Boucle(vCompteur;1;100)
  ` Faire quelque chose
Fin de boucle
```

Il est intéressant d'examiner la manière dont les boucles **Tant que...Fin tant que** et **Repeter...Jusque** effectuent la même action :

Voici la boucle **Tant que...Fin tant que** équivalente :

```
$i :=1 ` Initialisation du compteur
Tant que($i<=100) ` Boucle 100 fois
  ` Faire quelque chose
   $i :=$i +1 ` Il faut incrémenter le compteur
Fin tant que
```

Voici la boucle **Repeter...Jusque** équivalente :

```
$i :=1 ` Initialisation du compteur
Repeter
  ` Faire quelque chose
   $i :=$i +1 ` Il faut incrémenter le compteur
Jusque($i=100) ` Boucle 100 fois
```

Astuce : La boucle **Boucle...Fin de boucle** est généralement plus rapide que les boucles **Tant que...Fin tant que** et **Repeter...Jusque** car 4D teste la condition en interne pour chaque cycle de la boucle et incrémente lui-même le compteur. Par conséquent, nous vous conseillons de préférer à chaque fois que c'est possible la structure **Boucle...Fin de boucle**.

Optimiser l'exécution de Boucle...Fin de boucle

Vous pouvez utiliser comme compteur une variable interprocess, process ou locale, et lui attribuer le type Réel, Entier ou Entier long. Pour des boucles longues, et particulièrement en mode compilé, nous vous conseillons d'employer des variables locales de type Entier long.

(11) Voici un exemple :

```
C_ENTIER LONG($vlCompteur) ` Utilisons une variable locale de type Entier long
Boucle($vlCompteur;1;10000)
  ` Faire quelque chose
Fin de boucle
```

Structures Boucle...Fin de boucle emboîtées

Vous pouvez emboîter autant de structures répétitives que vous voulez (dans les limites du raisonnable). Cela s'applique aux structures de type **Boucle...Fin de boucle**. Il y a dans ce cas une erreur courante à éviter : assurez-vous d'utiliser une variable compteur différente par structure de boucle. Voici deux exemples :

(12) L'exemple suivant permet de traiter tous les éléments d'un tableau à deux dimensions :

```
Boucle($vlElem;1;Taille tableau(unTableau))
  ` ...
  ` Faire quelque chose avec la ligne
  ` ...
    Boucle($vlSousElem;1;Taille tableau(unTableau{$vlElem}))
  ` Faire quelque chose avec l'élément
      unTableau{$vlElem}{$vlSousElem}:=...
    Fin de boucle
Fin de boucle
```

(13) L'exemple suivant construit un tableau de pointeurs vers tous les champs de type Date présents dans la base :

```
TABLEAU POINTEUR($apChampsDate;0)
$vlElem:=0
Boucle($vlTable;1;Lire numero derniere table)
   Si(Est un numero de table valide($vlTable))
      Boucle($vlChamp;1;Lire numero dernier champ($vlTable))
         Si(Est un numero de champ valide($vlTable;$vlChamp))
            $vpChamp:=Champ($vlTable;$vlChamp)
            Si(Type($vpChamp>)=Est une date)
               $vlElem:=$vlElem+1
               INSERER DANS TABLEAU($apChampsDate;$vlElem)
               $apChampsDate{$vlElem}:=$vpChamp
            Fin de si
         Fin de si
      Fin de boucle
   Fin de si
Fin de boucle
```

Pour faire fonctionner les commandes, les opérateurs, et les autres composants du langage, vous les placez dans des méthodes. Ce chapitre décrit les fonctionnalités communes à tous les types de méthodes. Il existe plusieurs types de méthodes : les méthodes objet, les méthodes formulaire, les méthodes table (ou triggers), les méthodes projet et les méthodes base.

Une méthode est composée de plusieurs **lignes d'instructions**. Une ligne d'instructions effectue une action. Cette ligne d'instruction peut être simple ou complexe. Cette ligne peut être aussi longue que vous voulez (elle peut comporter jusqu'à 32 000 caractères, ce qui est normalement suffisant pour la plupart des instructions).

Par exemple, la ligne suivante est une instruction qui ajoute un nouvel enregistrement à la table [Personnes] :

```
AJOUTER ENREGISTREMENT([Personnes])
```

Une méthode contient également des **tests** et des **boucles** qui structurent son exécution. Pour plus d'informations sur les structures de programmation, reportez-vous à la section **Conditions et boucles**.

Note : La taille maximale d'une méthode est limitée à 2 Go de texte ou 32 000 lignes d'instructions. Au-delà de ces limites, un message d'alerte apparaît, indiquant que les lignes supplémentaires ne seront pas affichées.

Types de méthodes

Il existe cinq types de méthodes dans 4D :

- **Méthodes objet** : une méthode objet est une courte méthode associée à un objet actif dans un formulaire. En général, les méthodes objet "gèrent" l'objet au moment de l'affichage ou de l'impression du formulaire. Vous ne pouvez pas appeler une méthode objet, 4D l'exécute automatiquement lorsqu'un événement implique l'objet auquel la méthode est rattachée.
- **Méthodes formulaire** : Une méthode formulaire est associée à un formulaire. Vous pouvez utiliser une méthode formulaire pour gérer les données et les objets, mais il est généralement plus simple et plus efficace d'utiliser des méthodes objet dans ce cas. Vous ne pouvez pas appeler une méthode formulaire, 4D gère automatiquement son exécution lorsqu'un événement implique le formulaire auquel la méthode est attachée.
- **Méthodes table/triggers** : Un trigger est associé à une table. Vous ne pouvez pas appeler un trigger. 4D les exécute automatiquement à chaque opération élémentaire effectuée sur les enregistrements de la table (Ajout, Suppression et Modification). Les triggers sont des méthodes pouvant prévenir toute action "illégale" opérée sur les enregistrements. Par exemple, dans un système de facturation, vous pouvez empêcher les utilisateurs d'ajouter des factures sans avoir spécifié le client concerné. Les triggers constituent un outil très puissant pour contrôler les opérations effectuées sur les tables ainsi que pour prévenir toute perte de données accidentelle. Vous pouvez écrire des triggers très simples ou très sophistiqués.

Pour plus d'informations sur les triggers, reportez-vous à la section **Triggers**.

- **Méthodes projet** : A la différence des précédents types de méthodes, les méthodes projet ne sont associées à aucun objet, formulaire ou table, elles peuvent être utilisées à n'importe quel endroit de votre base. Les méthodes projet sont réutilisables et disponibles à tout moment, pour toute autre méthode. Si vous devez répéter certaines tâches, vous n'avez pas besoin de réécrire plusieurs méthodes identiques dans chaque cas. Vous pouvez appeler une méthode projet partout où vous en avez besoin — depuis d'autres méthodes projet, objet ou formulaire. Lorsque vous appelez une méthode projet, elle se comporte comme si vous l'aviez écrite à l'endroit d'où vous l'appelez. Les méthodes projet utilisées par d'autres méthodes sont appelées des **sous-routines**. Une méthode projet qui retourne un résultat peut aussi être appelée une **fonction**.

Il existe une autre manière d'appeler les méthodes projet : il suffit de les associer à des commandes de menus. Lorsque vous associez une méthode projet à une commande de menu, la méthode est exécutée lorsque la commande du menu est sélectionnée par un utilisateur.

- **Méthodes base** : Tout comme les méthodes formulaire ou objet sont exécutées lorsqu'un événement se produit dans un formulaire, il existe des méthodes associées à la base entière, et qui sont exécutées lorsqu'un événement de session de travail se produit. Ce sont les **méthodes base**. Par exemple, à chaque fois que vous ouvrez la base, vous pouvez vouloir initialiser des variables qui seront utilisées pendant toute la session de travail. Pour cela, vous pouvez écrire des instructions dans la **Méthode base Sur ouverture**, exécutée automatiquement par 4D lorsque vous lancez la base.

Pour plus d'informations sur les méthodes base, reportez-vous à la section **Méthodes base**.

Un exemple de méthode projet

Toutes les méthodes sont fondamentalement identiques — elles débutent sur la première ligne et traitent chaque ligne d'instruction jusqu'à ce qu'elles atteignent la dernière ligne (c'est-à-dire qu'elles s'exécutent séquentiellement). Voici un exemple de méthode projet :

```
CHERCHER([Personnes]) ` Afficher l'éditeur de recherches
Si(OK=1) ` L'utilisateur a cliqué sur OK (et non sur Annuler)
   Si(Enregistrements trouves([Personnes])=0) ` Si aucun enregistrement n'est trouvé…
      AJOUTER ENREGISTREMENT([Personnes])
 ` Permettre à l'utilisateur d'ajouter un enregistrement
   Fin de si
Fin de si ` Fin
```

Chaque ligne de l'exemple est une **ligne d'instruction**. Tout ce que vous écrivez dans le langage de 4D est appelé du code. Le code est exécuté — ce qui signifie que 4D effectue l'action spécifiée par le code.

Nous allons examiner très attentivement la première ligne ; nous irons plus rapidement par la suite :

```
CHERCHER([Personnes]) ` Afficher l'éditeur de recherches
```

Le premier mot de la ligne, **CHERCHER**, est une commande. Une commande est un élément du langage de 4D — elle effectue une action. Dans ce cas, **CHERCHER** affiche l'éditeur de recherches, tout comme le fait la commande **Chercher...** dans le menu **Enregistrements** en mode Développement.

Les parenthèses signifient qu'un **paramètre** est **passé** à la commande **CHERCHER**. Un paramètre (ou **argument**) est une valeur nécessaire à une commande pour remplir son rôle. Dans ce cas, *[Personnes]* est le nom d'une table. Les noms des tables sont toujours écrits entre crochets (*[…]*). Nous pouvons donc dire : « La table Personnes est un paramètre de la commande **CHERCHER** ». Une commande qui accepte plusieurs paramètres.

Enfin, il y a un **commentaire** à la fin de la ligne. Un commentaire vous informe (ainsi que toute personne qui examinera votre méthode) de ce qui se passe dans le code. Un commentaire est signalé par une apostrophe inversée (`). Dans une ligne, tout ce qui suit un signe de commentaire est ignoré lorsque le code est exécuté. Un commentaire peut être placé sur sa propre ligne, ou à la suite du code, comme dans l'exemple. N'hésitez pas à utiliser les commentaires dans vos méthodes ; la lecture et la compréhension de votre code sont facilitées, aussi bien pour vous-même que pour d'autres personnes.

Note : Un commentaire peut contenir jusqu'à 32 000 caractères.

La ligne suivante de notre exemple vérifie qu'aucun enregistrement n'a été trouvé :

```
Si(Enregistrements trouves([Personnes])=0) ` Si aucun enregistrement n'est trouvé…
```

La ligne d'instruction **Si** est un **test conditionnel** — une instruction qui contrôle l'exécution au pas à pas de votre méthode. Le **Si** effectue un test, et si le test est VRAI, la ou les lignes suivantes sont exécutées. **Enregistrements trouves** est une fonction — c'est-à-dire une commande qui retourne une valeur. Ici, **Enregistrements trouves** retourne le nombre d'enregistrements de la sélection courante de la table passée en paramètre.

Note : Seule la première lettre du nom d'une fonction est en majuscule. C'est la convention d'écriture utilisée pour les fonctions de 4D.Vous savez déjà ce qu'est la sélection courante : le groupe d'enregistrements avec lequel vous travaillez à un instant donné. Si le nombre d'enregistrements est égal à 0 (c'est-à-dire, si aucun enregistrement n'a été trouvé), la ligne de code suivante est exécutée :

```
AJOUTER ENREGISTREMENT([Personnes])
```

La commande **AJOUTER ENREGISTREMENT** affiche un formulaire pour permettre à l'utilisateur de créer un nouvel enregistrement. Notez que cette ligne comporte une indentation. 4D formate votre code automatiquement ; l'indentation vous permet d'identifier facilement ce qui est dépendant du test conditionnel (**Si**).

```
Fin de si ` Fin
```

La ligne d'instruction **Fin de si** conclut la séquence d'instructions contrôlée par le test **Si**. Pour chaque ligne d'instruction de test conditionnel, votre code doit comporter une instruction correspondante indiquant au langage où s'arrête le test. Nous vous conseillons de bien maîtriser les concepts évoqués dans ce chapitre. S'ils sont nouveaux pour vous, n'hésitez pas à relire les explications fournies jusqu'à ce qu'elles vous semblent claires.

Pour en savoir plus...

- Pour plus d'informations sur les méthodes objet et formulaire, reportez-vous à la description de la commande **Evenement formulaire** ainsi qu'au manuel Mode Développement de 4D.
- Pour travailler avec les triggers, reportez-vous à la section **Triggers**.
- Pour une description détaillée des méthodes projet, reportez-vous à la section **Méthodes projet**.
- Les méthodes base sont détaillées dans la section **Méthodes base**.

Méthodes projet

Les méthodes projet, comme leur nom l'indique, s'appliquent à la totalité de votre projet, c'est-à-dire votre base de données. Alors que les méthodes formulaire ou les méthodes objet par exemple sont associées à des formulaires ou des objets, une méthode projet est disponible partout — elle n'est associée à aucun élément particulier de la base. Une méthode projet peut tenir les rôles suivants, en fonction de la manière dont elle est exécutée et utilisée :

- Méthode de menu
- Sous-routine et fonction
- Méthode de gestion de process
- Méthode de gestion d'événements
- Méthode de gestion d'erreurs

Ces appellations ne qualifient pas la nature des méthodes (ce qu'elles sont), mais leur fonction (ce qu'elles font).

Une **méthode de menu** est une méthode projet appelée depuis une commande de menu personnalisé. Elle se comporte comme un agent de police chargé de la circulation, dirigeant les flux de votre application. Les méthodes de menu contrôlent, aiguillent lorsque c'est nécessaire, affichent les formulaires, génèrent des états ou encore gèrent votre base.

Le deuxième type de méthode projet peut être considéré comme une méthode asservie — d'autres méthodes lui demandent d'effectuer des tâches. Ce type de méthode est appelé **sous-routine**. Une sous-routine qui retourne une valeur est appelée une **fonction**.

Une **méthode de gestion de process** est une méthode projet appelée lorsqu'un process est démarré. Le process existera tant que la méthode sera en cours d'exécution. Pour plus d'informations sur les process, reportez-vous à la section **Process**. A noter qu'une méthode de menu associée à une commande de menu pour laquelle la propriété **Démarrer un nouveau process** est sélectionnée, est aussi la méthode de gestion de process pour le process créé.

Une **méthode de gestion d'événements** est une méthode dédiée à la gestion des événements, qui s'exécute dans un process différent de celui de la méthode de gestion des process. Généralement, pour la gestion des événements, vous pouvez laisser 4D faire le gros du travail. Par exemple, lors de la saisie de données, 4D détecte les clics souris et les touches enfoncées, puis appelle les méthodes objet et formulaire correspondantes, vous permettant ainsi de prévoir dans ces méthodes les traitements appropriés aux événements.
Dans d'autres circonstances, vous devez gérer directement les événements. Si, par exemple, vous exécutez une opération de longue durée (telle qu'une **Boucle...Fin de boucle** appliquée à chaque enregistrement), vous souhaitez pouvoir interrompre l'opération en tapant la touche **Esc**. Dans ce cas, une méthode de gestion d'événements est parfaitement adaptée. Pour plus d'informations, reportez-vous à la description de la commande **APPELER SUR EVENEMENT**.

Une **méthode de gestion d'erreurs** est une méthode projet d'interruption. Elle s'exécute à l'intérieur du process dans lequel elle a été installée à chaque fois qu'une erreur se produit. Pour plus d'informations, reportez-vous à la description de la commande **APPELER SUR ERREUR**.

Méthodes de menu

Une méthode de menu est appelée en mode Application lorsque la commande de menu personnalisé à laquelle elle est associée est sélectionnée. Vous assignez la méthode à la commande de menu dans l'éditeur de menus de 4D. Lorsque l'utilisateur sélectionne la commande de menu, la méthode est exécutée. Ce fonctionnement est l'un des principaux aspects de la personnalisation d'une base de données. C'est en créant des menus qui appellent des méthodes de menu que vous personnalisez votre base. Reportez-vous au manuel Mode Développement de 4D pour plus d'informations sur l'éditeur de menu.

Les commandes de menus personnalisés peuvent déclencher une ou plusieurs actions. Par exemple, une commande de menu de saisie d'enregistrements peut appeler une méthode effectuant deux actions : afficher le formulaire entrée approprié et appeler la commande **AJOUTER ENREGISTREMENT** jusqu'à ce que l'utilisateur annule la saisie de nouveaux enregistrements.

L'automatisation de séquences d'actions est une possibilité très puissante du langage de programmation de 4D. A l'aide des menus personnalisés, vous pouvez automatiser des séquences de tâches, vous permettez aux utilisateurs de naviguer plus facilement dans votre base.

Sous-routines

Lorsque vous avez écrit une méthode projet, elle devient partie intégrante du langage de la base dans laquelle elle a été créée. Vous pouvez alors l'appeler de la même manière que vous appelez les commandes intégrées de 4D. Une méthode projet utilisée de cette manière est appelée une sous-routine.

L'utilisation de sous-routines procure les avantages suivants :

- Réduction du code répétitif,
- Clarification des méthodes,
- Modification plus facile des méthodes,
- Création de code modulaire.

Imaginons par exemple que vous travaillez avec une base de clients. A mesure que vous construisez la base, vous vous apercevez que vous répétez souvent certaines tâches, telles que la recherche d'un client et la modification de son enregistrement. Le code nécessaire à l'accomplissement de cette opération pourrait être :

```
  ` Recherche d'un client
CHERCHER PAR EXEMPLE([Clients])
  ` Sélection du formulaire entrée
FORM FIXER ENTREE([Clients];"Saisie de données")
  ` Modification de l'enregistrement du client
MODIFIER ENREGISTREMENT([Clients])
```

Si vous n'utilisez pas de sous-routines, vous devrez écrire ce code à chaque fois que vous voudrez modifier l'enregistrement d'un client. Si cette opération peut être réalisée dans dix endroits différents de votre base, vous devrez la réécrire dix fois. Grâce aux sous-routines, vous ne l'écrirez qu'une seule fois en tout. C'est le premier avantage des sous-routines : réduire la quantité de code à écrire.

Si le code ci-dessus était une méthode projet appelée ***MODIFIER CLIENT***, vous l'exécuteriez simplement en inscrivant son nom dans une autre méthode. Par exemple, pour modifier l'enregistrement d'un client puis l'imprimer, vous n'auriez qu'à écrire :

```
MODIFIER CLIENT
IMPRIMER SELECTION([Clients])
```

Cette possibilité simplifie énormément vos méthodes. Dans l'exemple ci-dessus, il n'est pas nécessaire de savoir comment fonctionne la méthode ***MODIFIER CLIENT***, mais uniquement ce qu'elle fait. C'est le deuxième avantage que vous pouvez tirer de l'utilisation de sous-routines : la clarification de votre code. Ainsi, ces méthodes deviennent en quelque sorte des extensions du langage de 4D.

Si vous devez modifier votre mode de recherche des clients, comme dans notre exemple, il vous suffit de modifier une seule méthode, et non dix. C'est un autre avantage des sous-routines : faciliter les modifications de votre code.

Avec les sous-routines, vous rendez votre code modulaire. Cela signifie simplement que vous dissociez votre code en modules (sous-routines), chacun d'entre eux effectuant une tâche logique. Examinez le code suivant, tiré d'une base de gestion de comptes chèques :

```
CHERCHER CHEQUES EMIS ` Rechercher les chèques émis
RAPPROCHER COMPTE ` Rapprocher le compte
IMPRIMER RELEVE ` Imprimer un relevé
```

Même pour quelqu'un qui ne connaît pas la base, le code est clair. Il n'est pas nécessaire d'examiner chaque sous-routine. Elles peuvent contenir de nombreuses lignes d'instructions et effectuer des opérations complexes, mais l'important est ce qu'elles font.

Nous vous conseillons de découper votre code en tâches logiques, ou modules, à chaque fois que c'est possible.

Passer des paramètres aux méthodes

Vous aurez souvent besoin de fournir des valeurs à vos méthodes. Vous pouvez facilement effectuer cette opération grâce aux paramètres.

Les **paramètres** (ou arguments) sont des données dont les méthodes ont besoin pour s'exécuter — le terme "paramètres" ou "arguments" est utilisé indifféremment dans ce manuel. Des paramètres sont également passés aux commandes intégrées de 4D. Dans l'exemple ci-dessous, la chaîne "Bonjour" est un paramètre de la commande **ALERTE** :

```
ALERTE("Bonjour")
```

Les paramètres sont passés de la même manière aux méthodes. Par exemple, si la méthode ***FAIRE QUELQUE CHOSE*** accepte trois paramètres, l'appel à cette méthode pourrait être de la forme suivante :

```
FAIRE QUELQUE CHOSE(AvecCeci;EtCela;CommeCeci)
```

Les paramètres sont séparés par des points-virgules (*;*).

Dans la sous-routine (la méthode appelée), la valeur de chaque paramètre est automatiquement copiée séquentiellement dans des variables locales numérotées : *$1*, *$2*, *$3*, etc. La numérotation des variables locales représente l'ordre des paramètres.

Ces variables/paramètres locaux ne sont pas réellement les champs, variables ou expressions passées à la **méthode appelante** : elles contiennent simplement leurs valeurs.

A l'intérieur de la sous-routine, vous pouvez utiliser les paramètres *$1*, *$2...* de la même manière que vous utilisez les autres variables locales.

Note : Toutefois, dans le cas où vous utilisez des commandes qui modifient la valeur de la variable passée en paramètre (par exemple **Trouver dans champ**), les paramètres *$1*, *$2...* ne peuvent pas être utilisés directement. Vous devez d'abord les recopier dans des variables locales standard (par exemple *$mavar:=$1*).

Puisque ces variables sont locales, elles ne sont définies qu'à l'intérieur de la sous-routine et sont effacées à la fin de son exécution. Pour cette raison, une sous-routine ne peut pas modifier, au niveau de la méthode appelante, la valeur réelle des champs ou des variables passé(e)s en paramètre. Par exemple :

```
  ` Voici une partie de la méthode MA METHODE
  ` ...
FAIRE QUELQUE CHOSE([Personnes]Nom) ` Admettons que [Personnes]Nom est égal à "william"
ALERTE([Personnes]Nom)

  ` Voici le code de la méthode FAIRE QUELQUE CHOSE
$1:=Majusc($1)
ALERTE($1)
```

La boîte de dialogue d'alerte affichée par ***FAIRE QUELQUE CHOSE*** contiendra "WILLIAM" et celle affichée par ***MA METHODE*** contiendra "william". La méthode a modifié localement la valeur du paramètre *$1*, mais cela n'affecte pas la valeur du champ *[Personnes]Nom* passé en paramètre par la méthode ***MA METHODE***.

Si vous voulez réellement que la méthode ***FAIRE QUELQUE CHOSE*** modifie la valeur du champ, deux solutions s'offrent à vous :

1. Plutôt que de passer le champ à la méthode, vous lui passez un pointeur :

```
  ` Voici une partie de la méthode MA METHODE
  ` ...
FAIRE QUELQUE CHOSE(->[Personnes]Nom) ` Admettons que [Personnes]Nom est égal à "william"
ALERTE([Personnes]Nom)

  ` Voici le code de la méthode FAIRE QUELQUE CHOSE
$1->:=Majusc($1->)
ALERTE($1->)
```

Ici, le paramètre n'est pas le champ lui-même, mais un pointeur vers le champ. Ainsi, à l'intérieur de la méthode ***FAIRE QUELQUE CHOSE***, *$1* ne contient plus la valeur du champ mais un pointeur vers le champ. L'objet **référencé** par *$1* (*$1->* dans le code ci-dessus) est le champ lui-même. Par conséquent, la modification de l'objet référencé dépasse les limites de la sous-routine et le champ lui-même est affecté. Dans cet exemple, les deux boîtes de dialogue d'alerte afficheront "WILLIAM".

Pour plus d'informations sur les pointeurs, reportez-vous à la section **Pointeurs**.

2. Plutôt que la méthode ***FAIRE QUELQUE CHOSE*** "fasse quelque chose", vous pouvez la réécrire de manière à ce qu'elle retourne une valeur :

```
  ` Voici une partie de la méthode MA METHODE
  ` ...
[Personnes]Nom:=FAIRE QUELQUE CHOSE([Personnes]Nom) ` Admettons que [Personnes]Nom est égal à "william"
ALERTE([Personnes]Nom)

  ` Voici le code de la méthode FAIRE QUELQUE CHOSE
$0:=$1
ALERTE($0)
```

Une sous-routine retournant une valeur est appelée une fonction. Ce point est traité dans les paragraphes suivants.

Note de programmation avancée : Les paramètres à l'intérieur de la sous-routine sont accessibles par l'intermédiaire des variables locales *$1*, *$2*... De plus, des paramètres peuvent être optionnels et être référencés à l'aide de la syntaxe *${...}*. Pour plus d'informations sur ce point, reportez-vous à la description de la commande **Nombre de paramètres**.

Les fonctions, méthodes projet retournant une valeur

Les méthodes peuvent retourner des valeurs. Une méthode qui retourne une valeur est appelée une **fonction**.

Les commandes de 4D ou de plug-ins qui retournent une valeur sont également appelées fonctions.

Par exemple, la ligne d'instruction suivante utilise une fonction intégrée, **Longueur**, qui retourne la longueur d'une chaîne. La valeur retournée par **Longueur** est placée dans une variable appelée *MaLongueur* :

```
MaLongueur:=Longueur("Comment suis-je arrivé là ?")
```

Toute sous-routine peut retourner une valeur. La valeur à retourner est placée dans la variable locale *$0*.

Par exemple, la fonction suivante, appelée ***Majuscules4***, retourne une chaîne dont les quatre premiers caractères ont été passés en majuscules :

```
$0:=Majusc(Sous chaine($1;1;4))+Sous chaine($1;5)
```

Voici un exemple qui utilise la fonction ***Majuscules4*** :

```
NouvellePhrase:=Majuscules4("Bien joué.")
```

Dans ce cas, la variable **NouvellePhrase** prend la valeur "BIEN joué."

Le **retour de fonction**, *$0*, est une variable locale à la sous-routine. Elle peut être utilisée en tant que telle à l'intérieur de la sous-routine. Par exemple, dans le cas de la méthode ***FAIRE QUELQUE CHOSE*** utilisée précédemment, *$0* recevait d'abord la valeur de *$1*, puis était utilisée en tant que paramètre de la commande **ALERTE**. Dans une sous-méthode, vous pouvez utiliser *$0* comme n'importe quelle autre variable locale. C'est 4D qui retourne sa valeur finale (sa valeur courante au moment où la sous-routine se termine) à la méthode appelée.

Méthode projet récursives

Des méthodes projet peuvent s'appeler les unes les autres, et en particulier :

- Une méthode A peut appeler une méthode B, qui appelle A, donc A appelle B de nouveau, etc.
- Une méthode peut s'appeler elle-même.

Cela s'appelle la **récursivité**. Le langage de 4D supporte pleinement la récursivité.

Examinons l'exemple suivant : vous disposez d'une table *[Amis et relations]* composée de l'ensemble de champs suivant (très simplifié) :
- *[Amis et parents]Nom*
- *[Amis et parents]Enfant'Nom*

Pour cet exemple, nous supposons que les valeurs des champs sont uniques (il n'existe pas deux personnes avec le même nom). A partir d'un nom, vous voulez écrire la phrase "Un de mes amis, Pierre, qui est le rejeton de Paul qui est le rejeton de Martine qui est le rejeton de Robert qui est le rejeton de Gertrude, fait cela pour gagner sa vie !" :

1. Vous pouvez procéder de la manière suivante :

```
$vsNom:=Demander("Saisissez le nom :";"Pierre")
Si(OK=1)
   CHERCHER([Amis et parents];[Amis et parents]Nom=$vsNom)
   Si(Enregistrements trouves([Amis et parents])>0)
      $vtHistoireComplète:="Un de mes amis, "+$vsNom
      Repeter
         CHERCHER([Amis et parents];[Amis et parents]Enfant'Nom=$vsNom)
         $vlResultRecherche:=Enregistrements trouves([Amis et parents])
         Si($vlResultRecherche>0)
            $vtHistoireComplète:=$vtHistoireComplète+" qui est le rejeton de "+[Amis et parents]Nom
            $vsNom:=[Amis et parents]Nom
         Fin de si
      Jusque($vlResultRecherche=0)
      $vtHistoireComplète:=$vtHistoireComplète+", fait cela pour gagner sa vie !"
      ALERTE($vtHistoireComplète)
   Fin de si
Fin de si
```

2. Vous pouvez également procéder ainsi :

```
$vsNom:=Demander("Saisissez le nom :";"Pierre")
```

```
Si(OK=1)
   CHERCHER([Amis et parents];[Amis et parents]Nom=$vsNom)
   Si(Enregistrements trouves([Amis et parents])>0)
      ALERTE("Un de mes amis, "+Généalogie de($vsNom)+", fait cela pour gagner sa vie !")
   Fin de si
Fin de si
```

en utilisant la fonction récursive ***Généalogie de*** suivante :

```
  ` Méthode projet Généalogie de
  ` Généalogie de ( Chaîne ) -> Texte
  ` Généalogie de ( Nom ) -> Partie de la phrase

$0:=$1
CHERCHER([Amis et parents];[Amis et parents]Enfant'Nom=$1)
Si(Enregistrements trouves([Amis et parents])>0)
   $0:=$0+" qui est le rejeton de "+Généalogie de([Amis et parents]Nom)
Fin de si
```

Vous notez que la méthode ***Généalogie de*** s'appelle elle-même.

La première manière de procéder utilise un **algorithme itératif**. La seconde manière utilise un **algorithme récursif**.

Lorsque vous implémentez du code pour traiter des cas comme celui décrit ci-dessus, vous aurez toujours le choix entre écrire des méthodes utilisant des algorithmes itératifs ou récursifs. Généralement, la récursivité permet d'écrire du code plus concis, lisible et plus facilement modifiable, mais utiliser la récursivité n'est absolument pas obligatoire.

Dans 4D, la récursivité est typiquement utilisée pour :

- Traiter les enregistrements de tables liées les unes aux autres de la même manière que décrit dans l'exemple ci-dessus.
- Naviguer parmi les documents et les dossiers de votre disque à l'aide des commandes **LISTE DES DOSSIERS** et **LISTE DES DOCUMENTS**. Un dossier peut contenir des dossiers et des documents, les sous-dossiers peuvent eux-mêmes contenir des dossiers et des documents, etc.

Important : Les appels récursifs doivent toujours se terminer à un moment donné. Dans l'exemple ci-dessus, la méthode ***Généalogie de*** cesse de s'appeler elle-même lorsque la recherche ne trouve plus d'enregistrement. Sans ce test conditionnel, la méthode s'appellerait indéfiniment et 4D pourrait au bout d'un certain temps retourner l'erreur “La pile est pleine” car le programme n'aurait plus assez de place pour "empiler" les appels (ainsi que les paramètres et les variables locales utilisés dans la méthode).

Débogueur

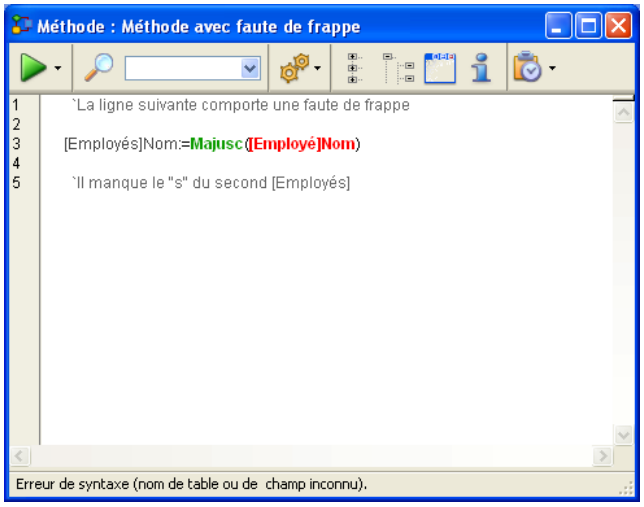
Un débogueur, pour quoi faire ?

Lorsque vous développez et testez vos méthodes, il est important que vous puissiez repérer et corriger les erreurs qui peuvent s'y être glissées.

Vous pouvez commettre plusieurs types d'erreurs en utilisant le langage : des fautes de frappe, des erreurs de syntaxe ou liées à l'environnement, des erreurs logiques ou de conception, ou encore des erreurs d'exécution (Runtime).

Fautes de frappe

Les fautes de frappe sont détectées directement dans l'**éditeur de méthodes** et sont indiquées par un libellé de couleur rouge ainsi qu'un message dans la zone d'information située en bas de la fenêtre. Cette fenêtre signale une faute de frappe :



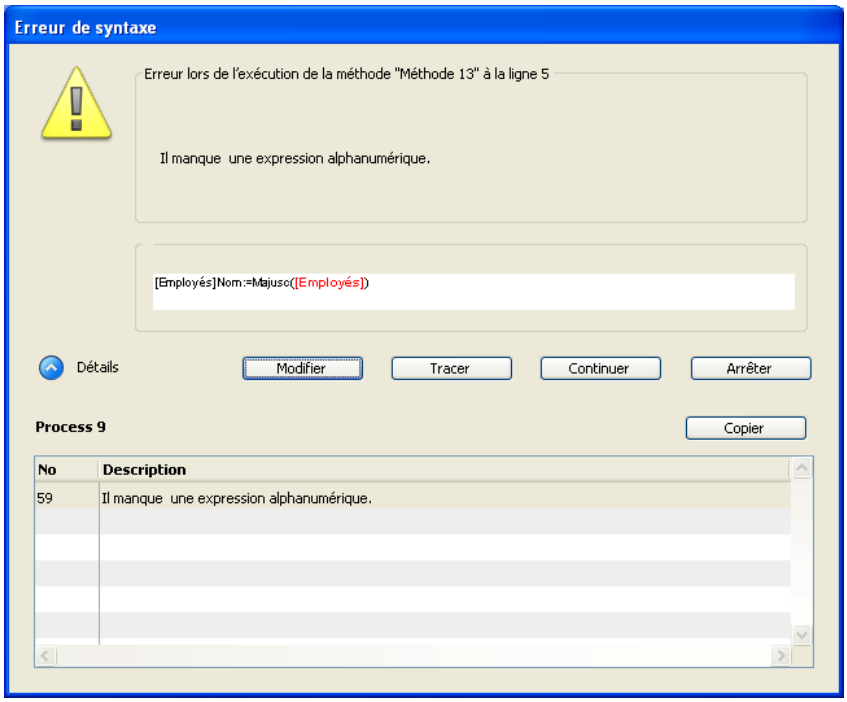
Note : Les commentaires ont été manuellement ajoutés pour cet exemple. Seule la couleur est modifiée par 4D à l'endroit de l'erreur.

De telles fautes de frappe provoquent généralement des erreurs de syntaxe (dans ce cas, le nom de la table est inconnu). La zone d'informations de la fenêtre affiche la cause de l'erreur au moment de la validation de la ligne d'instructions.

Vous pouvez alors corriger l'erreur de frappe et appuyer sur la touche **Entrée** (du clavier numérique) pour valider la correction. Pour plus d'informations sur l'éditeur de méthodes, reportez-vous au manuel Mode Développement de 4D.

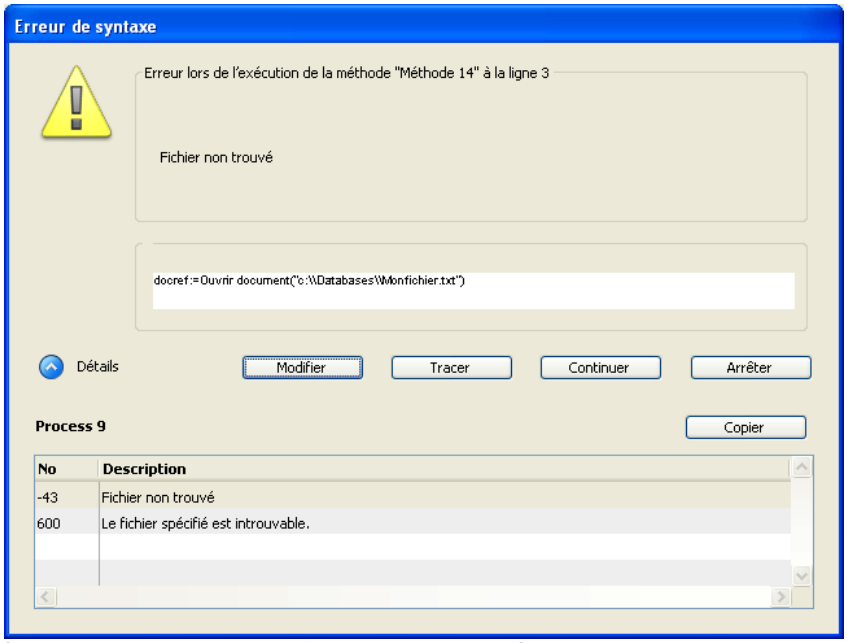
Erreurs de syntaxe ou liées à l'environnement

Certaines erreurs ne peuvent être détectées que lorsque vous exécutez la méthode. La **Fenêtre d'erreur de syntaxe** est affichée lorsqu'une telle erreur est détectée. Par exemple :



Ici, l'erreur provient du fait que le nom d'une table est passé à la commande **Majusc**, qui attend une expression de type Texte. Dans cette image, la zone "Détail" est déployée afin d'afficher la dernière erreur et son numéro.

Parfois, il peut arriver que vous n'ayez pas assez de mémoire pour créer un tableau ou un BLOB. Ou bien, lorsque vous cherchez à accéder à un document sur votre disque, ce document peut ne pas exister ou être déjà ouvert par une autre application.



Ces erreurs ne se produisent pas à cause de votre code. Simplement, ce sont des choses qui arrivent ! La plupart d'entre elles sont faciles à identifier à l'aide d'une méthode d'interception d'erreurs (reportez-vous à la description de la commande **APPELER SUR ERREUR**.)

Pour plus d'informations sur cette fenêtre, reportez-vous à la section **Fenêtre d'erreur de syntaxe**.

Erreurs logiques ou de conception

Ce sont généralement les erreurs qui sont les plus difficiles à repérer. Vous devez utiliser le débogueur pour les détecter. Notez qu'à l'exception des fautes de frappe, tous les types d'erreurs indiqués précédemment sont aussi, parfois, des erreurs de logique ou de conception. Voici des exemples :

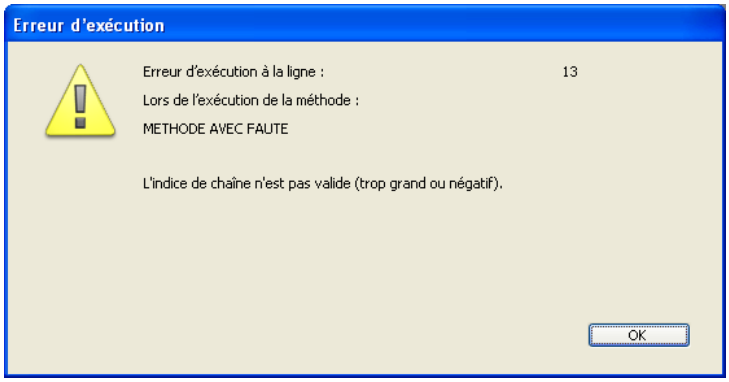
- Une erreur de syntaxe peut être retournée si vous essayez d'utiliser une variable qui n'est pas encore initialisée.
- Une erreur liée à l'environnement peut se produire si vous essayez d'ouvrir un document dont le nom est reçu par une sous-routine qui ne reçoit pas la bonne valeur dans le paramètre. Notez que dans cet exemple, le morceau de code qui ne fonctionne pas n'est pas celui qui est à l'origine du problème.

Les erreurs logiques ou de conception se produisent également dans les situations suivantes :

- Un enregistrement n'est pas correctement mis à jour parce qu'en appelant **STOCKER ENREGISTREMENT**, vous avez oublié de tester d'abord si cet enregistrement était ou non verrouillé.
- Une méthode ne fait pas exactement ce que vous attendiez parce que vous ne testez pas la présence éventuelle d'un paramètre optionnel.

Erreurs d'exécution

En mode Application, vous pouvez obtenir des erreurs que vous n'avez jamais vues en mode interprété. Voici un exemple :



Ce message vous indique que vous essayez d'accéder à un caractère dont la position se trouve au-delà de la longueur de la chaîne. Pour trouver rapidement l'origine du problème, notez le nom de la méthode et le numéro de la ligne, ouvrez votre base en mode interprété puis allez à la méthode et à la ligne indiquées.

Que faire lorsque vous rencontrez une erreur ?

Les erreurs sont chose commune. Il est rare d'écrire une grande quantité de lignes de code sans générer d'erreur. Traiter et corriger les erreurs est tout aussi naturel.

Grâce à son environnement multitâche, 4D vous permet de modifier et d'exécuter rapidement vos méthodes : vous n'avez qu'à passer d'une fenêtre à une autre. De plus, vous découvrirez combien il est simple et rapide de repérer vos erreurs en utilisant le **Débogueur**.

Un réflexe courant chez les débutants lorsqu'une erreur est détectée est de cliquer sur le bouton **Arrêter** dans la fenêtre d'erreur de syntaxe, de retourner à l'éditeur de méthodes, et d'essayer de deviner ce qui se passe en regardant le code. Ne faites pas cela ! Vous gagnerez un temps considérable en utilisant **toujours** le **Débogueur**.

- S'il se produit une erreur de syntaxe inattendue, utilisez le **Débogueur**.
- S'il se produit une erreur d'environnement, utilisez le **Débogueur**.
- S'il se produit tout autre type d'erreur, utilisez le **Débogueur**.

Dans 99 % des cas, le **Débogueur** affiche l'information dont vous avez besoin pour comprendre la cause de l'erreur. Vous pouvez alors la corriger.

Astuce : Passez quelques heures à apprendre et expérimenter le **Débogueur**. Vous gagnerez des journées et des mois dans l'avenir.

La phase de développement représente une autre occasion d'utiliser le **Débogueur**. Imaginons que vous deviez écrire un algorithme plus complexe que d'habitude. Une fois le code écrit, vous ne pouvez être certain qu'il fonctionne tant que vous ne l'aurez pas testé. Au lieu de passer directement en exécution, vous pouvez d'abord le vérifier en insérant la commmande **TRACE** au début de votre code. Ensuite, exécutez le code au pas à pas pour contrôler ce qui se passe.

Conclusion générale

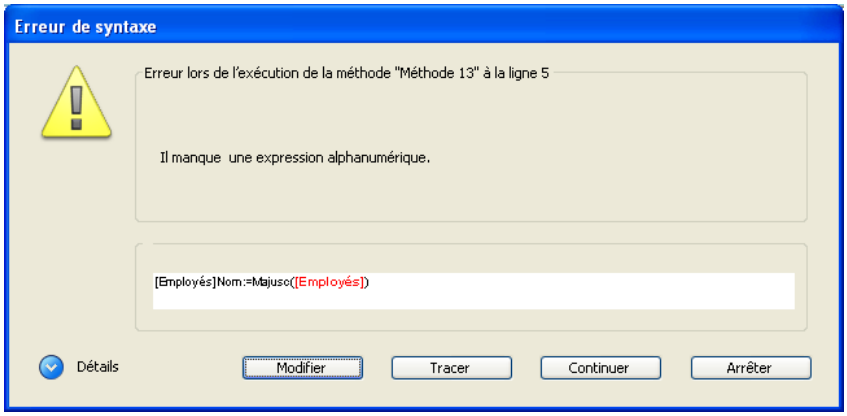
Utilisez le **Débogueur**.

Fenêtre d'erreur de syntaxe

La **Fenêtre d'erreur de syntaxe** s'affiche lorsque l'exécution d'une méthode est interrompue. L'exécution de la méthode peut être interrompue pour l'une des raisons suivantes :

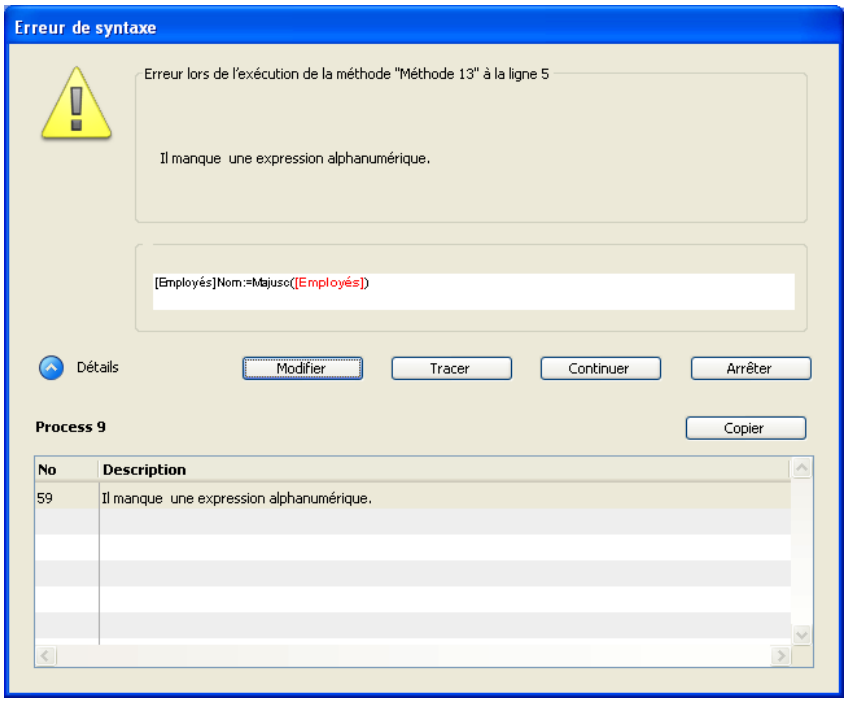
- 4D interrompt la méthode car une erreur l'empêche de poursuivre son exécution.
- La méthode a produit une assertion fausse (cf. commande **ASSERT**).

Voici une fenêtre d'erreur de syntaxe :



Le texte situé dans la zone supérieure de la fenêtre affiche un message décrivant l'erreur. La partie inférieure fait apparaître la ligne exécutée au moment où l'erreur est survenue ; l'emplacement précis où est survenue l'erreur est sélectionné.

Le bouton **Détails** permet de déployer la partie inférieure de la fenêtre affichant la "pile" d'erreurs liées au process :



La fenêtre comporte cinq boutons : **Arrêter**, **Tracer**, **Continuer**, **Modifier** et (si la fenêtre est déployée) **Copier**.

- **Arrêter** : La méthode est interrompue et vous retournez à l'endroit où vous vous trouviez avant de commencer l'exécution de la méthode. Si une méthode formulaire ou une méthode objet s'exécutent en réponse à un événement, elles sont stoppées et vous retournez au formulaire. Si la méthode s'exécute à partir du mode Application, vous retournez dans ce mode.

- **Tracer** : Vous entrez dans le mode Trace et la fenêtre du **Débogueur** est affichée. Si la ligne courante a été partiellement exécutée, il se peut que vous soyez obligé de cliquer plusieurs fois sur le bouton **Tracer**. Lorsque la ligne est terminée, la fenêtre du **Débogueur** s'affiche.

- **Continuer** : L'exécution continue. La ligne contenant l'erreur peut avoir été partiellement exécutée — tout dépend de l'endroit où se trouvait l'erreur. Continuez avec prudence — l'erreur peut empêcher que le reste de la méthode s'exécute correctement. Généralement, il vaut mieux ne pas continuer. Vous pouvez cliquer sur **Continuer** si l'erreur se trouve dans un appel mineur, comme par exemple **CHANGER TITRE FENETRE**, qui n'empêche pas de continuer l'exécution et le test du code. Vous pouvez vous concentrer sur le code le plus important, et corriger les erreurs mineures ultérieurement.
  Note : Si vous appuyez sur la touche **Alt** (Windows) ou **Option** (Mac OS), le libellé du bouton **Continuer** devient **Ignorer**. Cliquer sur le bouton **Ignorer** permet de ne plus afficher la fenêtre si la même erreur, déclenchée par la même méthode à la même ligne, survient à nouveau. Ce raccourci est utile dans le cas d'une erreur se produisant de façon répétitive, par exemple dans une boucle. Tout se passe dans ce cas comme si vous cliquiez à chaque fois sur le bouton **Continuer**.

- **Modifier** : L'exécution de la méthode est totalement interrompue. 4D passe en mode Développement. La méthode dans laquelle l'erreur est survenue est ouverte dans l'éditeur de méthodes, ce qui vous permet de la corriger. Utilisez cette option lorsque vous avez identifié immédiatement l'erreur et

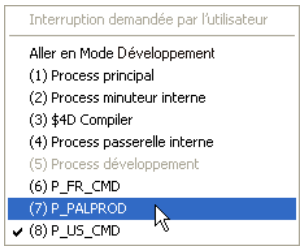
que vous pouvez la corriger sans qu'il soit nécessaire d'effectuer d'autres investigations.

- **Copier** : Ce bouton provoque la copie des informations de débogage dans le Presse-papiers. Ces informations décrivent l'environnement interne de l'erreur (numéro, composant interne, etc.). Elles sont formatées en texte tabulé. Une fois que vous avez cliqué sur ce bouton, vous pouvez coller le contenu du Presse-papiers dans un fichier texte, un tableur, un message email, etc. à des fins d'analyse.

Le mot Débogueur provient du terme anglais bug (insecte ou punaise) qui se "traduit" en français par bogue . Une bogue dans une méthode est une erreur que vous désirez éliminer. Lorsqu'une erreur se produit dans votre code, ou lorsque vous désirez contrôler l'exécution de vos méthodes, vous utilisez le débogueur. Un débogueur vous permet de trouver les bogues en exécutant pas à pas vos méthodes et en fournissant toutes les informations nécessaires. Ce procédé s'appelle le **mode Trace**.

Pour appeler la fenêtre du débogueur et afficher puis tracer les méthodes, vous pouvez :

- Cliquer sur le bouton **Tracer** dans la **Fenêtre d'erreur de syntaxe**,
- Utiliser la commande **TRACE**,
- Cliquer sur le bouton **Déboguer** dans la fenêtre d'exécution de méthode.
- Utiliser la combinaison **Alt+Maj+clic droit** (sous Windows) ou **Control+Option+Commande+clic** (sous Mac OS) pendant l'exécution d'une méthode, puis choisir le process à tracer dans le pop up menu qui apparaît :

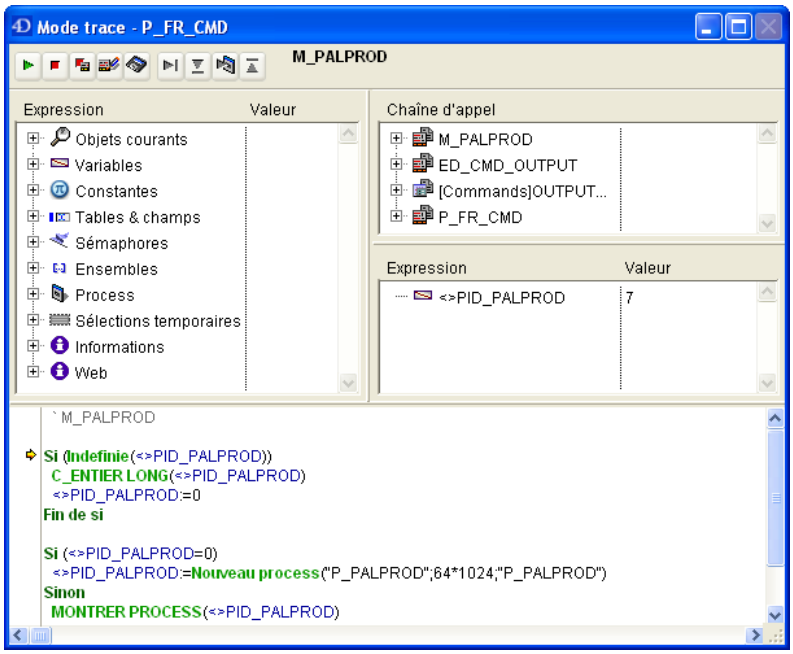


- Cliquer sur le bouton **Tracer** en mode Développement, lorsqu'un process est sélectionné dans la page **Process** de l'Explorateur d'exécution.
- Créer ou modifier un point d'arrêt dans la fenêtre d'édition d'une méthode, ou dans les pages **Point d'arrêt** et **Arrêt sur commande** de l'Explorateur d'exécution.

Note : Lorsque vous tracez un process en cours d'exécution, la fenêtre du débogueur s'affiche instantanément. Si vous tracez un process qui n'est pas en cours d'exécution (process endormi, en attente d'événement, etc...), le débogueur "enregistre" la requête et n'apparaîtra qu'au moment où le process aura repris l'exécution du code.

Note : Si votre base de données est dotée d'un système de mots de passe, seuls le Super_Utilisateur et les utilisateurs possédant les privilèges d'accès au développement peuvent tracer des méthodes.

Voici la fenêtre du débogueur :

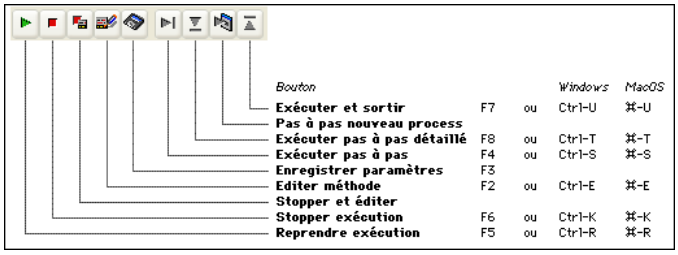


Vous pouvez déplacer la fenêtre du débogueur et/ou redimensionner toutes les zones internes de la fenêtre comme vous le souhaitez. L'affichage ultérieur d'une nouvelle fenêtre du débogueur reprend la configuration (taille et position de la fenêtre, emplacement des lignes de division et contenu de la zone d'évaluation des expressions) de la dernière fenêtre affichée dans la même session.

4D est un environnement multitâche. Dans le cas de plusieurs process s'exécutant simultanément, vous pouvez tracer chacun d'entre eux de manière indépendante. Chaque process peut avoir sa propre fenêtre de débogueur.

Barre d'outils de contrôle d'exécution

La **barre d'outils de contrôle d'exécution**, située en haut de la fenêtre du débogueur, comporte neuf boutons. Voici leur description ainsi que leurs raccourcis clavier associés :



Bouton 'Reprendre exécution'

Arrêt du mode Trace et reprise du cours normal de l'exécution de la méthode.

Note : La combinaison **Maj+F5** ou **Maj+clic** sur le bouton **Reprendre exécution** provoque la reprise de l'exécution avec désactivation de tous les appels à **TRACE** suivants dans le process courant.

Bouton 'Stopper exécution'

La méthode s'arrête et vous retournez là où vous étiez avant son exécution. Si vous étiez en train de tracer une méthode formulaire ou une méthode objet s'exécutant en réponse à un événement, elle s'arrête et vous retournez au formulaire. Si vous traciez une méthode s'exécutant à partir du mode Application, vous retournez à ce mode.

Bouton 'Stopper et éditer'

La méthode s'arrête comme lorsque vous cliquez sur **Stopper exécution**. De plus, 4D affiche dans la fenêtre de l'éditeur de méthodes la méthode qui s'exécutait au moment où vous avez cliqué sur le bouton.

Conseil : Utilisez ce bouton lorsque vous connaissez les modifications à apporter à votre code, et le moment où elles doivent être effectuées pour pouvoir poursuivre le test de vos méthodes. Une fois vos modifications effectuées, ré-exécutez la méthode.

Bouton 'Editer méthode'

Ce bouton se comporte comme le bouton **Stopper et éditer**, à la différence près qu'il n'annule pas l'exécution en cours. L'exécution de la méthode est simplement suspendue à l'instant du clic sur le bouton. 4D affiche dans la fenêtre de l'éditeur de méthodes la méthode qui s'exécutait au moment où vous avez cliqué sur le bouton **Editer méthode**.

Important : Vous pouvez modifier cette méthode, mais ces modifications n'apparaîtront pas, ou ne s'exécuteront pas dans l'instance de la méthode tracée dans la fenêtre du débogueur. Ce n'est qu'une fois que la méthode aura été stoppée ou entièrement exécutée que les modifications pourront apparaître. En d'autres termes, il faut recharger la méthode pour que les modifications soient prises en compte.

Conseil : Utilisez ce bouton lorsque vous connaissez les modifications à apporter à votre code et lorsqu'elles n'interfèrent pas avec le reste du code qui doit être exécuté ou tracé.

Astuce : Les méthodes objet sont rechargées pour chaque événement. Si vous tracez une méthode objet (par exemple en réponse à un clic sur un bouton) vous n'avez pas besoin de quitter le formulaire. Vous pouvez modifier la méthode objet, sauvegarder les modifications, puis retourner au formulaire et tester à nouveau. Pour tracer/modifier les méthodes formulaire, il faut sortir du formulaire puis le réouvrir pour recharger la méthode correspondante. L'astuce est donc la suivante : lorsque vous effectuez le débogage complet d'un formulaire, placez le code que vous déboguez dans une méthode projet que vous utiliserez comme sous-routine à l'intérieur de la méthode formulaire. De cette manière, vous pouvez rester dans le formulaire, tracer, modifier et tester à nouveau votre formulaire car la sous-routine sera rechargée chaque fois qu'elle sera appelée par la méthode formulaire.

Bouton 'Enregistrer paramètres'

Ce bouton permet de sauvegarder la configuration courante de la fenêtre du débogueur (taille et position de la fenêtre, emplacement des lignes de division et contenu de la zone d'évaluation des expressions). Elle sera alors utilisée par défaut à chaque ouverture de la base.
Ces paramétrages sont stockés dans le fichier de structure de la base.

Bouton 'Exécuter pas à pas'

La ligne courante de la méthode (indiquée par la flèche jaune — cette flèche s'appelle le **compteur de programme**) est exécutée et le débogueur passe à la ligne suivante. Le bouton **Exécuter pas à pas** ne passe pas dans les sous-routines et les fonctions. Il reste au niveau de la méthode que vous êtes en train de tracer. Si vous voulez également tracer les appels aux sous-routines et aux fonctions, utilisez le bouton **Pas à pas détaillé**.

Bouton 'Pas à pas détaillé'

Lors de l'exécution d'une ligne qui appelle une autre méthode (sous-routine ou fonction), ce bouton provoque l'affichage de la méthode appelée dans la fenêtre du débogueur, et permet au développeur de passer pas à pas dans cette méthode. La nouvelle méthode devient la méthode courante (en haut) dans la sous-fenêtre **Fenêtre de chaîne d'appel** de la fenêtre du débogueur. Lors de l'exécution d'une ligne qui n'appelle pas une autre méthode, ce bouton se comporte comme le bouton **Exécuter pas à pas**.

Bouton 'Pas à pas nouveau process'

Lors de l'exécution d'une ligne qui crée un nouveau process (par exemple qui appelle la commande **Nouveau process**) ce bouton ouvre une nouvelle fenêtre du débogueur qui vous permet de tracer la méthode de gestion du process que vous venez de créer. Lors de l'exécution d'une ligne qui ne crée pas de nouveau process, ce bouton se comporte comme le bouton **Exécuter pas à pas**.

Bouton 'Exécuter et sortir'

Si vous tracez des sous-routines et des fonctions, cliquer sur ce bouton vous permet d'exécuter l'intégralité de la méthode qui est en train d'être tracée, et de revenir à la méthode appelante. La fenêtre du débogueur retourne à la méthode précédente dans la chaîne d'appel. Si la méthode courante est la dernière méthode de la chaîne d'appel, la fenêtre du débogueur se referme.

Informations dans la barre d'outils de contrôle d'exécution

A la droite de la barre d'outils de contrôle d'exécution, le débogueur affiche les informations suivantes :

- Le nom de la méthode que vous êtes en train de tracer (en noir).
- La cause de l'ouverture du débogueur (en rouge).

Par exemple, dans la fenêtre affichée en tête de ce chapitre, vous pouvez voir les informations suivantes :

- La méthode ***Trace démo*** est la méthode qui est tracée.
- La fenêtre du débogueur s'affiche car il a rencontré un point d'arrêt.

Voici les causes qui provoquent l'apparition du débogueur et d'un message (en rouge) :

- **Commande TRACE** : un appel à **TRACE** a été émis.
- **Point d'arrêt atteint** : vous avez rencontré un point d'arrêt.
- **Interruption demandée par l'utilisateur** : vous avez activé **Alt**+**Maj**+**clic droit** (sous Windows) ou **Control**+**Option**+**Commande**+**clic** (sous Mac OS) ou encore cliqué sur le bouton **Trace** dans la page **Process** de l'Explorateur d'exécution.
- **Détection d'un appel à : Nom de la commande** : Un appel à une commande 4D qui doit être interceptée est sur le point d'être exécuté.
- **Pas à pas nouveau process** : Vous avez cliqué sur le bouton **Pas à pas nouveau process** et ce message est affiché par la fenêtre du débogueur ouverte pour le process qui vient d'être créé.

Les sous-fenêtres du débogueur

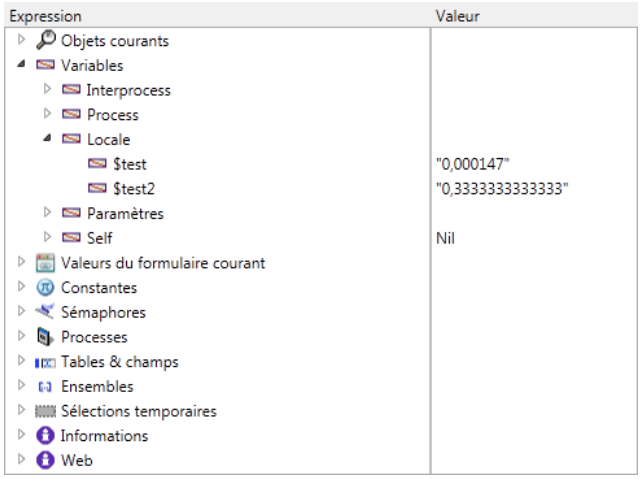
La fenêtre du débogueur contient la barre de contrôle d'exécution décrite précédemment, ainsi que quatre sous-fenêtres redimensionnables :

- **Fenêtre d'expression**
- **Fenêtre de chaîne d'appel**
- **Fenêtre d'évaluation**
- **Fenêtre d'évaluation des méthodes**

Les trois premières fenêtres affichent des listes hiérarchiques montrant l'information pertinente au débogage. La quatrième, la **Fenêtre d'évaluation des méthodes**, affiche le code source de la méthode tracée. Chaque fenêtre a un rôle précis pour vous assister dans le processus de débogage. Avec la souris, vous pouvez redimensionner la fenêtre du débogueur, ainsi que chaque sous-fenêtre. En outre, les trois premières sous-fenêtres sont séparées par une ligne pointillée dans le sens de la hauteur : vous pouvez redimensionner à votre convenance les colonnes à l'intérieur de chaque sous-fenêtre.

Fenêtre d'expression

La **Fenêtre d'expression** est située en haut à gauche de la fenêtre du débogueur, sous la barre d'outils de contrôle d'exécution :



La **Fenêtre d'expression** affiche toutes les informations générales utiles sur l'environnement système, 4D et l'exécution du code.

La colonne **Expression** affiche le nom des objets et des expressions. La colonne **Valeur** affiche la valeur courante correspondant aux objets et expressions.

En cliquant sur une valeur dans la colonne de droite, vous pouvez modifier la valeur de l'objet, si cela est possible pour cet objet.

La liste hiérarchique multi-niveaux est organisée par thèmes au niveau supérieur. Les thèmes sont les suivants :

- Objets courants
- Variables
- Valeurs du formulaire courant
- Constantes
- Sémaphores
- Process
- Tables & champs
- Ensembles
- Sélections temporaires
- Informations
- Web

En fonction du thème, chaque article peut disposer d'un ou de plusieurs sous-niveaux. Pour déployer ou fermer chaque thème, il suffit de cliquer sur l'icône de déploiement située en face de son libellé. Si le thème comprend plusieurs niveaux d'information, il suffit de cliquer sur chaque icône pour explorer toutes les informations qu'il contient.

A tout moment, vous pouvez faire glisser un thème, un sous-thème ou un article, et le déposer dans la **Fenêtre d'évaluation**.

Objets courants

Ce thème affiche les valeurs des objets ou des expressions évaluables :

- utilisé(e)s dans la ligne de code à exécuter (celle qui est indiquée par le compteur de programme — la flèche jaune dans la **Fenêtre d'évaluation des méthodes**),
- utilisé(e)s dans la ligne de code précédente.

Comme la ligne de code précédente est celle qui vient d'être exécutée, le thème Objets courants affiche donc de manière permanente les objets ou expressions de la ligne courante **avant et après** l'exécution de la ligne. Prenons l'exécution de la méthode suivante :

```
TRACE
a:=1
b:=a+1
c:=a+b
` ...
```

1. Vous entrez dans la fenêtre du débogueur avec le compteur de programme de la **Fenêtre d'évaluation des méthodes** placé à la ligne *a:=1*. A ce point précis, le thème Objets courants affiche :

a: Indéfini

La variable *a* est affichée car elle est utilisée dans la ligne qui est sur le point d'être exécutée (mais elle n'a pas encore été initialisée).

2. Vous progressez d'une ligne. Le compteur de programme est maintenant positionné sur la ligne *b:=a+1*. A ce point, le thème Objets courants affiche :

a: *1*
b: Indéfini

La variable *a* s'affiche car elle est utilisée dans la ligne qui vient de s'exécuter, et qu'on lui avait assigné la valeur numérique 1. Elle s'affiche aussi parce

qu'elle est utilisée dans la ligne qui sera exécutée, en tant qu'expression qui sera assignée à la variable *b*. La variable *b* s'affiche car elle est utilisée dans la ligne qui doit être exécutée (mais elle n'a pas encore été initialisée).

3. Vous progressez encore d'une ligne. Le compteur de programme est maintenant positionné sur la ligne *c:=a+b*. A ce point, le thème Objets courants affiche :

c:	Indéfini
a:	*1*
b:	*2*

La variable *c* s'affiche car elle est utilisée dans la ligne à exécuter (mais elle n'a pas encore été initialisée). Les variables *a* et *b* sont affichées car elles étaient utilisées dans la ligne précédente et qu'elles le sont dans la ligne qui sera exécutée, etc.

Ce thème est extrêment pratique : chaque fois que vous exécutez une ligne, vous n'avez pas besoin de saisir une expression dans la fenêtre d'évaluation. Il vous suffit de surveiller les valeurs affichées par le thème Objets courants.

Variables

Ce thème se décompose en sous-thèmes :

- **Interprocess** : affiche la liste des variables interprocess en cours d'utilisation. Si vous n'utilisez pas de variable interprocess, la liste est vide. La valeur des variables interprocess peut être modifiée.
- **Process** : affiche la liste des variables process utilisées par le process courant. Cette liste est rarement vide. La valeur des variables process peut être modifiée.
- **Locales** : affiche la liste des variables locales utilisées par la méthode indiquée dans la sous-fenêtre d'évaluation de méthode. Cette liste peut être vide si aucune variable locale n'est utilisée ou si elles n'ont pas encore été créées. La valeur des variables locales peut être modifiée.
- **Paramètres** : affiche la liste des paramètres reçus par la méthode. Cette liste peut être vide si aucun paramètre n'a été passé à la méthode tracée. La valeur des paramètres peut être modifiée.
- **Self Pointeur** : affiche un pointeur vers l'objet courant si vous tracez une méthode objet. Cette valeur n'est pas modifiable.

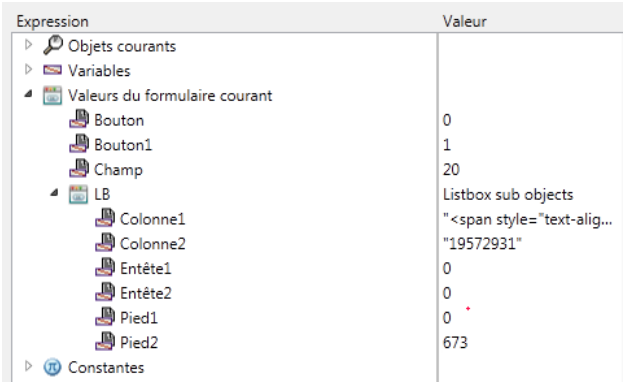
Note : Vous pouvez modifier les variables et les champs de type Chaîne, Texte, numérique (Entier, Entier long, Réel), Date et Heure, c'est-à-dire les variables et les champs dont la valeur peut être saisie au clavier. Pour le type Entier vous pouvez utiliser indifféremment la notation décimale ou hexadécimale (par exemple, pour 256, vous pouvez taper '256' ou '0x100').

Les tableaux, comme les autres variables, apparaissent dans les sous-thèmes Interprocess, Process et Locales, en fonction de leur portée. Le débogueur affiche chaque tableau avec un niveau hiérarchique supplémentaire, ce qui vous permet d'obtenir ou de modifier les valeurs des éléments du tableau, s'il y en a. Le débogueur affiche les 100 premiers éléments (y compris l'élément zéro). La colonne Valeur affiche la taille du tableau en face de son nom. Une fois que vous avez déployé le tableau, le premier sous-article affiche le numéro de l'élément sélectionné courant, ensuite l'élément zéro, ensuite les autres éléments (jusqu'à 100) s'il y en a. Vous pouvez modifier les tableaux Alpha, Texte, Numérique et Date. Vous pouvez modifier le numéro de l'élément sélectionné, l'élément zéro et les autres éléments (jusqu'à 100) s'il y en a. Vous ne pouvez pas modifier la taille du tableau.

Note : A tout moment, vous pouvez glisser un article à partir de la **Fenêtre d'expression** vers la **Fenêtre d'évaluation**, y compris un élément de tableau.

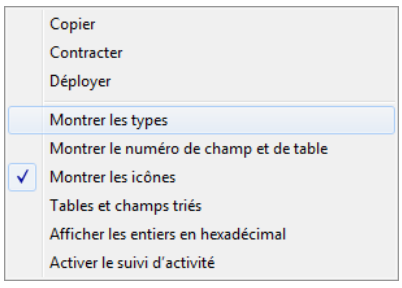
Valeurs du formulaire courant

Ce thème contient le nom de chaque objet dynamique présent dans le formulaire courant, ainsi que la valeur de sa variable associée :

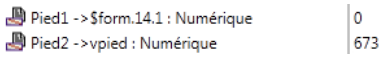


Certains objets, comme les list box tableaux, peuvent être représentés sous forme de deux objets (la variable de l'objet lui-même et sa source de données).

Cette liste est particulièrement utile lorsque votre formulaire utilise de nombreuses variables dynamiques : il est alors facile d'identifier les variables dynamiques par l'intermédiaire de leurs noms d'objets. Vous pouvez afficher le nom interne des variables dynamiques en sélectionnant la commande **Montrer les types** dans le menu contextuel :



Les noms des variables dynamiques sont de la forme "$form.4B9.42" :



Constantes

Ce thème affiche les constantes prédéfinies dans 4D, comme dans la page Constantes de la fenêtre de l'Explorateur. Les expressions de ce thème ne peuvent pas être modifiées.

Tables et champs

Ce thème affiche la liste des tables et des champs dans la base de données (à l'exception des sous-champs). Pour chaque table, la colonne Valeur affiche la taille de la sélection courante pour le process courant ainsi que (lorsque la ligne de la table est déployée) le nombre d'enregistrements verrouillés. Pour chaque champ, la colonne Valeur affiche la valeur du champ (à l'exception des images, sous-tables et BLOBs) pour l'enregistrement courant, s'il existe. Dans ce thème, les valeurs des champs peuvent être modifiées (notez qu'il n'y a alors pas d'annulation possible) mais pas celles de la table.

Sémaphores

Ce thème affiche la liste des sémaphores locaux dans les ensembles courants. Pour chaque sémaphore, la colonne Valeurs affiche le nom du process ayant posé le sémaphore. Si vous n'utilisez pas de sémaphore, la liste peut être vide. Les expressions de ce thème ne peuvent pas être modifiées. Il n'est pas possible de visualiser les sémaphores globaux.

Ensembles

Ce thème affiche la liste des ensembles définis dans le process courant (celui que vous êtes en train de tracer) ainsi que la liste des ensembles interprocess. La colonne Valeur affiche, pour chaque ensemble, le nombre d'enregistrements et le nom de la table. Si vous n'utilisez pas les ensembles, cette liste sera vide. Les expressions de ce thème ne peuvent pas être modifiées.

Process

Ce thème affiche la liste des process lancés depuis le début de la session de travail. La colonne Valeur affiche le temps déjà alloué à chaque process ainsi que son état (par exemple "En cours d'exécution", "Endormi", etc). Les expressions de ce thème ne peuvent pas être modifiées.

Sélections temporaires

Ce thème affiche la liste des sélections temporaires process définies dans le process courant (celui que vous êtes en train de tracer) ainsi que la liste des sélections temporaires interprocess. Pour chaque sélection temporaire, la colonne Valeur affiche le nombre d'enregistrements et le nom de la table. Si vous n'utilisez pas les sélections temporaires, cette liste sera vide. Les expressions de ce thème ne peuvent pas être modifiées.

Informations

Ce thème affiche des informations générales relatives au fonctionnement de la base, telles que la table par défaut courante (s'il y en a une), la mémoire physique, virtuelle, libre, occupée, la destination de recherche, etc. Ces informations permettent d'étudier le fonctionnement de la base.

Web

Ce thème affiche des informations relatives au serveur Web de l'application (informations disponibles uniquement si le serveur Web est actif) :

- Fichier Web à envoyer : nom du fichier Web en attente d'envoi (le cas échéant)
- Occupation du cache Web : nombre de pages présentes dans le cache Web et pourcentage d'utilisation,
- Temps d'activité du serveur Web : durée de fonctionnement au format heures:minutes:secondes du serveur Web
- Nombre de requêtes http : nombre total de requêtes HTTP reçues depuis le démarrage du serveur Web, ainsi que nombre instantané de requêtes par seconde
- Nombre de process Web actifs : nombre de process Web actifs, tous types de process Web confondus.

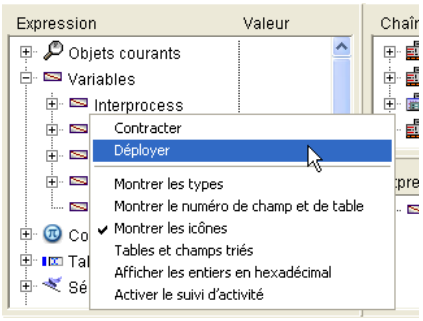
Les expressions contenues dans ce thème ne peuvent pas être modifiées.

Menu contextuel de la fenêtre d'expression

Le menu contextuel de la fenêtre d'expression vous propose des options supplémentaires. Pour afficher ce menu il vous suffit de :

- Sous Windows, cliquer n'importe où dans la fenêtre d'expression avec le **bouton droit** de la souris
- Sous Mac OS, utiliser la combinaison **Control+clic** n'importe où dans la fenêtre d'expression.

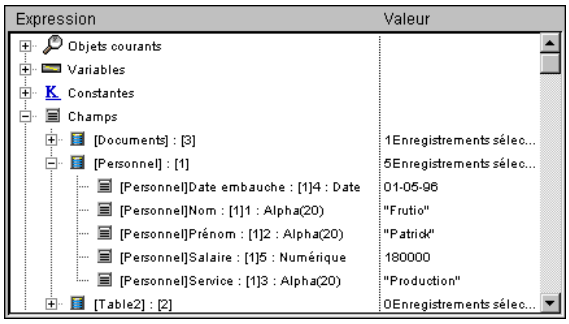
Voici le menu contextuel de la fenêtre d'expression :



- **Contracter** : Contracte tous les niveaux de la liste hiérarchique des expressions.
- **Déployer** : Déploie tous les niveaux de la liste hiérarchique des expressions.
- **Montrer les types** : Lorsque vous sélectionnez cette option, le type de l'objet s'affiche en face de l'objet (lorsque cela est pertinent).
- **Montrer le numéro de champ et de table** : Si vous travaillez avec le numéro des tables ou de champs, ou avec des pointeurs utilisant les commandes **Table** ou **Champ**, cette option est très pratique : en face de chaque table et champ, elle affiche le numéro de la table ou du champ.

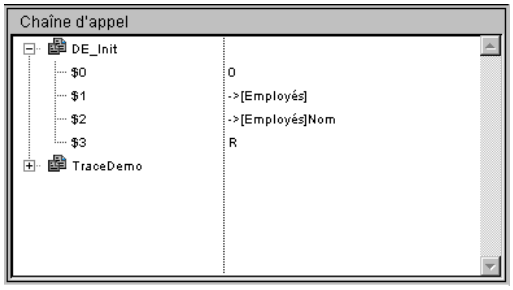
- **Montrer les icônes** : Chaque objet est précédé d'une icône qui indique son type. Vous pouvez désactiver cette option pour accélérer l'affichage, ou tout simplement parce que l'option **Montrer les types** vous convient.
- **Tables et champs triés** : Cette option force les tables et les champs à s'afficher par ordre alphabétique (dans leurs listes respectives).
- **Afficher les entiers en hexadécimal** : Les nombres s'affichent en notation décimale. Sélectionnez cette option pour les afficher en hexadécimal. **Note** : Pour exprimer une valeur numérique en hexadécimal, saisissez 0x (zéro + "x") puis les caractères hexadécimaux.
- **Activer le suivi d'activité** : Active le suivi d'activité (contrôle avancé de l'activité interne de l'application) et affiche les informations collectées dans des thèmes supplémentaires, **Séquenceur**, **Web** et **Réseau**.

Ci-dessous, la fenêtre d'expression telle qu'elle se présente lorsque vous sélectionnez toutes les options :



Fenêtre de chaîne d'appel

Une méthode peut appeler d'autres méthodes, qui à leur tour peuvent appeler d'autres méthodes. Pour cette raison, il est très utile d'avoir sous les yeux, pendant le débogage, la chaîne des méthodes, ou **chaîne d'appel**. Cette chaîne peut être visualisée dans la fenêtre située en haut et à droite du débogueur. Les méthodes y sont affichées de manière hiérarchique :



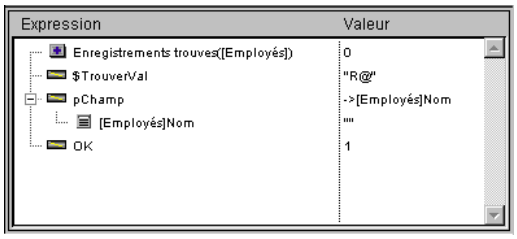
- Chaque niveau principal est le nom d'une méthode. L'élément placé en tête de la liste est la méthode que vous êtes en train de tracer, le niveau suivant est le nom de la méthode appelante (la méthode qui a appelé celle que vous êtes en train de tracer), le niveau suivant est l'appelant de la méthode appelante, etc. Dans l'exemple ci-dessus, la méthode **DE_Init** est tracée. Elle a été appelée par la méthode **TraceDemo**.
- Lorsque vous double-cliquez sur le nom d'une méthode dans la fenêtre de chaîne d'appel, vous basculez sur la méthode appelante dont le code source est affiché dans la fenêtre d'évaluation de méthodes. Vous pouvez ainsi voir rapidement comment la méthode appelante a effectué son appel à la méthode appelée. Vous pouvez aussi examiner toutes les étapes de la chaîne d'appel.
- Lorsque vous cliquez sur l'icône de déploiement jouxtant le nom d'une méthode, vous déployez ou vous contractez la liste des paramètres (*$1, $2...*) ainsi que le résultat (*$0*) optionnel d'une fonction. La valeur s'affiche à droite de la fenêtre. En cliquant sur une valeur quelconque à droite, vous pouvez changer la valeur du résultat ou de tout paramètre. Dans l'illustration ci-dessus :
  1. **DE_Init $0** est actuellement indéfinie car la méthode n'a assigné aucune valeur à *$0* (parce qu'elle n'a pas encore exécuté cette affectation, ou parce que la méthode est une sous-routine et non une fonction).
  2. **DE_Init** a reçu trois paramètres de **TraceDemo**. *$1* est un pointeur vers la table *[Employés]*, *$2* est un pointeur vers le champ *[Employés]Nom* et *$3* est un paramètre alphanumérique de valeur "**R**".
- Lorsque vous avez déployé la liste des paramètres/résultats d'une méthode, vous pouvez également les faire glisser vers la **Fenêtre d'évaluation**.

Fenêtre d'évaluation

Juste au-dessous de la **Fenêtre de chaîne d'appel** se trouve la **Fenêtre d'évaluation**. Cette fenêtre sert à évaluer les expressions. Vous pouvez évaluer tout type d'expression, y compris les champs, les variables, les pointeurs, les calculs, les fonctions intégrées, vos propres fonctions, et tout ce qui retourne une valeur.

Vous pouvez évaluer toute expression qui peut être affichée sous forme de texte. Cela ne s'applique donc pas aux champs ni aux variables image ou BLOB. En revanche, lorsque vous déployez les listes hiérarchiques, le débogueur vous permet d'afficher les tableaux et les pointeurs. Pour afficher le contenu des BLOBs, vous pouvez utiliser les commandes sur les BLOBs, comme par exemple **BLOB vers texte**.

Dans l'exemple ci-dessous, vous pouvez voir plusieurs expressions : deux variables, un pointeur vers un champ, le résultat d'une fonction intégrée et un calcul.



Insérer une nouvelle expression

Vous pouvez ajouter une expression à évaluer dans la fenêtre d'expression de l'une des manières suivantes :

- Glissez-déposez un objet ou une expression à partir de la **Fenêtre d'expression** ;
- Glissez-déposez un objet ou une expression à partir de la **Fenêtre de chaîne d'appel** ;
- Cliquez sur une expression évaluable dans la **Fenêtre d'évaluation des méthodes** .

En outre, pour créer une expression vide, double-cliquez n'importe où dans l'espace vide de la fenêtre d'évaluation. Cela crée une expression vide ` *Nouvelle expression* prête à l'édition. Vous pouvez saisir toute formule 4D qui retourne un résultat.

Dès que vous avez saisi la formule, appuyez sur la touche **Entrée** ou **Retour chariot** (ou cliquez n'importe où dans la fenêtre) pour obtenir l'évaluation de l'expression.

Si vous voulez changer d'expression, cliquez dessus pour la sélectionner, et cliquez de nouveau pour entrer en mode édition.

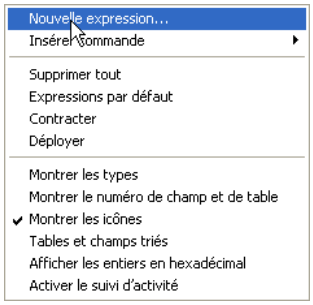
Si vous n'avez plus besoin d'une expression, cliquez dessus pour la sélectionner, puis appuyez sur la touche **Retour Arrière** ou **Suppr**.

Attention : Soyez prudent lorsque vous évaluez une expression 4D modifiant la valeur d'une des **Variables système** (par exemple la variable OK). En effet, dans ce cas l'exécution du reste de la méthode peut être faussée.

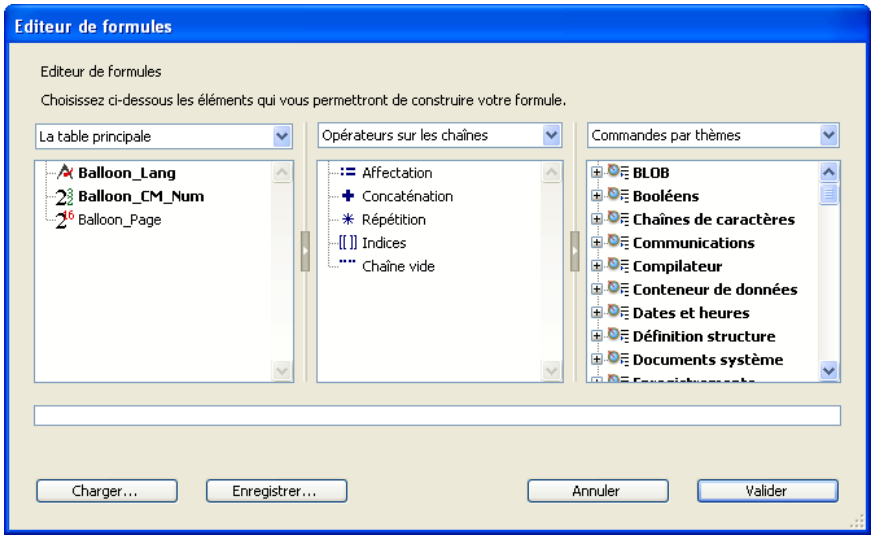
Menu contextuel de la fenêtre d'évaluation

Le menu contextuel de la **Fenêtre d'évaluation** vous permet d'accéder à l'éditeur de formules de 4D, pour vous aider à saisir et éditer une expression. Le menu contextuel vous propose également des options supplémentaires.

Pour obtenir ce menu, cliquez n'importe où dans la fenêtre d'évaluation avec le **bouton droit** de la souris.



- **Nouvelle expression** : Cette commande insère une nouvelle expression et affiche l'éditeur de formules de 4D (voir ci-dessous) dans lequel vous pouvez saisir la nouvelle expression.



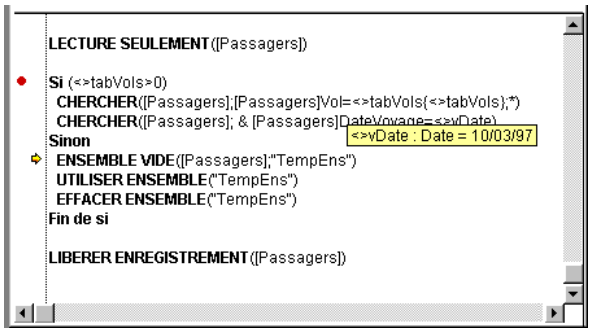
Pour plus d'informations sur l'éditeur de formules, reportez-vous au manuel Mode Développement de 4D.

- **Insérer commande** : Cette commande est un raccourci pour placer une commande (sans utiliser l'éditeur de formules) en tant que nouvelle expression.
- **Supprimer tout** : Efface toutes les expressions présentes.
- **Expressions par défaut** : Recopie la liste d'objets de la zone Expression.
- **Contracter/Déployer** : Contracte ou déploie toutes les expressions dont l'évaluation se fait en utilisant les listes hiérarchiques (pointeurs, tableaux, etc.).
- **Montrer les types** : Si cette option est sélectionnée, le type de l'objet s'affiche en face de chaque objet (lorsque c'est pertinent).
- **Montrer le numéro de champ et de table** : Cette commande est extrêmement pratique si vous travaillez avec des numéros de table ou de champs, ou avec des pointeurs utilisant des commandes comme **Table** ou **Champ**. En face de chaque table et champ, elle affiche le numéro de la table ou du champ.
- **Montrer les icônes** : Chaque objet est précédé par une icône qui symbolise son type. Vous pouvez désactiver cette option pour accélérer l'affichage, ou simplement parce que l'option **Montrer les types** vous convient.
- **Tables et champs triés** : Cette option force l'affichage des tables et des champs par ordre alphabétique (dans leurs listes respectives).
- **Afficher les entiers en hexadécimal** : Par défaut, les nombres entiers sont affichés en notation décimale. Sélectionnez cette option si vous souhaitez un affichage hexadécimal.
- **Activer le suivi d'activité** : Active et affiche les informations de suivi d'activité (cf. **Fenêtre d'expression**).

Fenêtre d'évaluation des méthodes

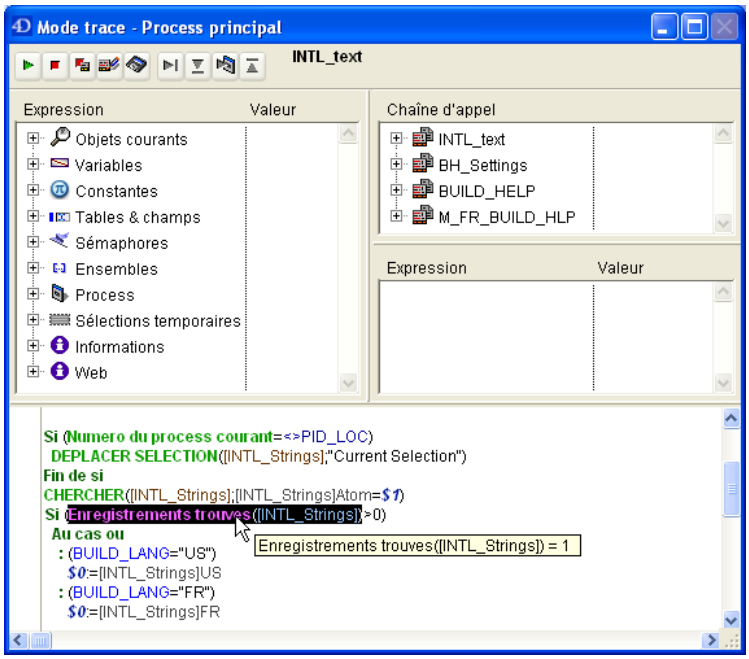
La **Fenêtre d'évaluation des méthodes** affiche le code source de la méthode en train d'être tracée.

- Si la méthode est trop longue pour tenir dans la zone de texte, vous pouvez la faire défiler.
- Lorsque vous positionnez le pointeur de la souris au-dessus d'une expression qui peut être évaluée (champ, variable, pointeur, tableau,...) le débogueur affiche une **Info-bulle** qui indique la valeur courante de l'objet ou de l'expression, et son type déclaré. Par exemple :



Une info-bulle apparaît lorsque le pointeur de la souris est positionné sur la variable *<>vDate*. Dans cet exemple, c'est une variable de type date contenant la valeur suivante : *10/03/97*.

- Vous pouvez également sélectionner une portion de texte dans la zone affichant le code en cours d'exécution. Dans ce cas, lorsque le curseur de la souris est placé au-dessus du texte sélectionné, l'info-bulle fournit la valeur de la sélection :



Lorsque vous cliquez sur un nom de variable ou de champ, il est automatiquement sélectionné.

Astuce : Vous pouvez copier instantanément dans la **Fenêtre d'évaluation** l'expression ou l'objet sélectionné. Vous disposez de deux possibilités :

- par simple glisser-déposer : cliquez sur le texte sélectionné, faites-le glisser et relâchez-le dans la zone d'évaluation.
- utiliser la combinaison de touches **Ctrl+D** (Windows) ou **Commande+D** (Mac OS).

Compteur de programme

Une flèche jaune, dans la marge gauche de la fenêtre d'évaluation des méthodes (voir illustration ci-dessus) indique la prochaine ligne à être exécutée. Cette flèche s'appelle le **compteur de programme**. Elle indique toujours la ligne sur le point d'être exécutée.

Pendant le débogage, vous pouvez **déplacer** le compteur de programme. Il vous suffit de cliquer sur la flèche jaune et de la faire glisser devant la ligne que vous désirez.

ATTENTION : Utilisez cette fonction avec précaution ! Déplacer vers l'avant le compteur de programme ne signifie pas que le débogueur exécute en même temps les lignes sur lesquelles vous passez. De la même manière, le faire remonter ne signifie pas que le débogueur inverse l'effet des lignes qui ont déjà été exécutées.

Lorsque vous déplacez le compteur de programme, vous indiquez simplement au débogueur de "continuer à tracer et exécuter à partir de cet endroit ". Tous les paramètres, champs, variables, etc. courants, ne sont pas affectés par le déplacement.

Voici un exemple de déplacement du compteur de programme. Imaginons que vous êtes en train de déboguer le code suivant :

```
...
Si(Cette condition)
   FAIRE QUELQUE CHOSE
```

```
Sinon
   FAIRE AUTRE CHOSE
Fin de si
```

Le compteur de programme est placé à la ligne **Si(Cette condition)**. Vous avancez d'un pas, et voyez que le compteur se place à la ligne ***FAIRE AUTRE CHOSE***. Pas de chance, car vous vouliez en fait exécuter la première partie de l'alternative. Dans ce cas, et dans la mesure où l'expression **Cette condition** n'effectue pas d'opérations affectant les étapes suivantes de votre test, remontez le compteur à la ligne ***FAIRE QUELQUE CHOSE***, et vous êtes prêt à continuer à tracer la portion de code qui vous intéresse.

Définir des points d'arrêt dans le débogueur

Pendant le débogage, vous pouvez avoir besoin de sauter certaines portions de votre code. Le débogueur vous permet d'utiliser plusieurs méthodes pour exécuter votre code **jusqu'à un certain point** :

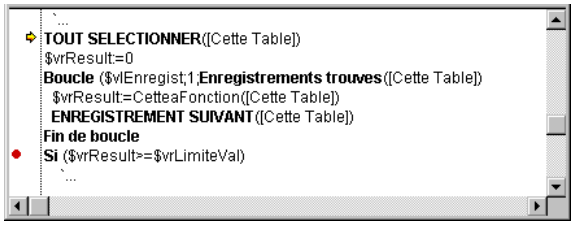
- Pendant que vous exécutez au pas à pas, vous pouvez cliquer sur le bouton **Exécuter pas à pas** au lieu de **Pas à pas détaillé**, lorsque vous ne voulez pas entrer dans les éventuelles sous-routines ou fonctions appelées dans la ligne du compteur de programme.
- Si vous entrez par erreur dans une sous-routine, vous pouvez l'exécuter et revenir directement à la méthode appelante en cliquant sur le bouton **Exécuter et sortir**.
- Si vous appelez la commande **TRACE** quelque part, vous pouvez cliquer sur le bouton **Pas de Trace** pour reprendre l'exécution jusqu'à cet appel **TRACE**.

Maintenant, imaginons que vous soyez en train d'exécuter le code suivant. Le compteur de programme est placé devant la ligne **TOUT SELECTIONNER ([CetteTable])** :

```
 //...
TOUT SELECTIONNER([CetteTable])
$vrResult:=0
Boucle($vlEnregist;1;Enregistrements trouves([CetteTable]))
   $vrResult:=CetteaFonction([CetteTable])
   ENREGISTREMENT SUIVANT([CetteTable])
Fin de boucle
Si($vrResult>=$vrLimiteVal)
 // ...
```

Vous voulez évaluer la valeur de *$vrResult* à la fin de la boucle **Boucle**. Comme cela peut prendre un certain temps d'exécution pour atteindre cette portion de votre code, vous voulez abandonner l'exécution courante puis éditer la méthode pour insérer un appel **TRACE** avant la ligne **Si ($vrResult...**

Vous pourriez aussi exécuter la boucle, mais si la table *[CetteTable]* contient plusieurs centaines d'enregistrements, cette opération vous prendra la journée. Pour éviter des situations de ce type, le *débogueur* met à votre disposition des **points d'arrêt**. Vous insérez des points d'arrêt en cliquant dans la marge gauche de la fenêtre d'évaluation des méthodes.
Par exemple, vous cliquez dans la marge gauche de la fenêtre, au niveau de la ligne **Si ($vrResult...** :



Vous insérez un point d'arrêt à la ligne marquée par un point rouge. Ensuite, vous cliquez sur le bouton **Pas de Trace**.
Vous reprenez l'exécution normale **jusqu'à** la ligne marquée par le point d'arrêt. Cette ligne n'est pas exécutée : vous êtes revenu au mode Trace. Dans cet exemple, toute la boucle a été exécutée normalement, et, une fois arrivé au point d'arrêt, il vous suffit de placer le curseur de la souris au-dessus de *vrResult* pour évaluer sa valeur à la sortie de la boucle.
Si vous placez un point d'arrêt au-delà du compteur de programme et que vous cliquez sur le bouton **Pas de Trace**, vous éviterez de tracer des parties de la méthode.

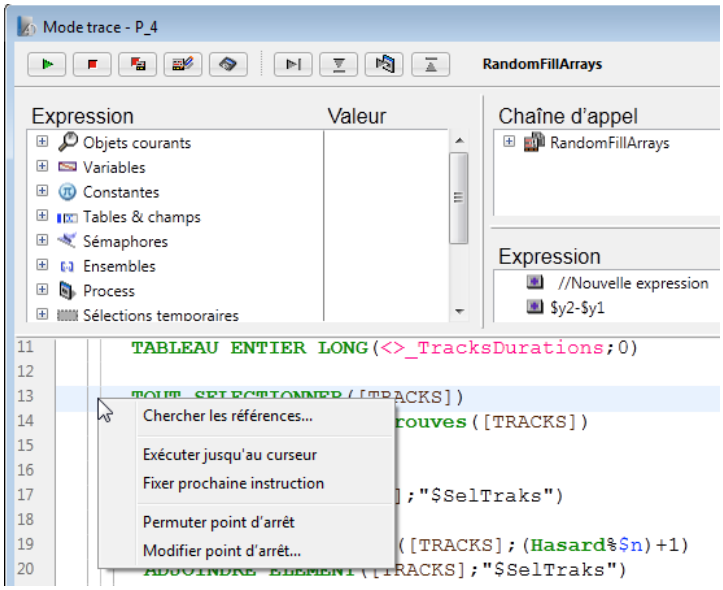
Note : Vous pouvez également définir des points d'arrêt directement dans l'éditeur de méthodes de 4D. Reportez-vous à la section **Points d'arrêt**.

Une fois que vous avez ajouté un point d'arrêt, il reste associé à la méthode, même lorsque vous quittez la base et la réouvrez par la suite. Pour le supprimer, vous pouvez procéder d'une des manières suivantes :

- Si vous avez fini de l'utiliser, cliquez dessus, et le point d'arrêt disparaît.
- Si vous n'avez pas fini de l'utiliser, mais que vous voulez le désactiver provisoirement, il vous faut l'éditer. Reportez-vous pour ceci à la section **Points d'arrêt**.

Menu contexuel de la fenêtre d'évaluation des méthodes

Le menu contextuel de la **Fenêtre d'évaluation des méthodes** donne accès à plusieurs fonctions utiles en phase d'exécution des méthodes en mode Trace :

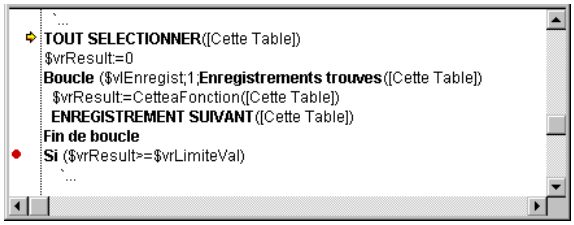


- **Aller à définition :** permet d'accéder à la définition de l'objet sélectionné. Cette commande est disponible avec les objets suivants :
  - *méthode projet* : affiche le contenu de la méthode dans une nouvelle fenêtre de l'éditeur de méthodes.
  - *champ* : affiche les propriétés du champ dans l'inspecteur de la fenêtre de structure,
  - *table* : affiche les propriétés de la table dans l'inspecteur de la fenêtre de structure,
  - *formulaire* : affiche le formulaire dans l'éditeur de formulaires,
  - *variable* (locale, process, interprocess ou paramètre $n) : affiche la ligne de déclaration de la variable dans la méthode courante ou parmi les méthodes compilateur.
- **Chercher les références** : cette fonction est également accessible depuis l'éditeur de méthodes. Elle permet de rechercher tous les objets de la base (méthodes et formulaires) dans lesquels l'élément courant de la méthode est référencé. L'élément courant est l'élément sélectionné ou l'élément dans lequel se trouve le curseur. Il peut s'agir d'un nom de champ, de variable, de commande, d'une chaîne, etc. Le résultat de la recherche est affiché dans une nouvelle fenêtre de résultat standard.
- **Exécuter jusqu'au curseur** : provoque l'exécution des instructions situées entre le compteur de programme (flèche jaune) et la ligne sélectionnée de la méthode (dans laquelle se trouve le curseur).
- **Fixer prochaine instruction** : déplace le compteur de programme jusqu'à la ligne sélectionnée sans l'exécuter et sans exécuter les lignes intermédiaires. La ligne désignée ne sera exécutée que si l'utilisateur clique sur l'un des boutons d'exécution.
- **Permuter point d'arrêt** : cette fonction est également accessible depuis l'éditeur de méthodes. Elle permet alternativement d'insérer ou de supprimer le point d'arrêt correspondant la ligne sélectionnée. Cette fonction modifie le point d'arrêt de façon permanente : par exemple, un point d'arrêt supprimé dans le débogueur n'apparaît plus dans la méthode d'origine.
- **Modifier point d'arrêt...** : cette fonction est également accessible depuis l'éditeur de méthodes. Elle permet d'afficher la boîte de dialogue de définition des Propriétés du point d'arrêt. Cette fonction modifie le point d'arrêt de façon permanente.

Comme décrit dans la section **Fenêtre d'évaluation des méthodes**, vous définissez un point d'arrêt en cliquant dans la marge gauche de la fenêtre d'évaluation des méthodes ou de la fenêtre d'édition des méthodes, au niveau de la ligne de code sur laquelle vous voulez vous arrêter.

Note : L'insertion et l'édition de points d'arrêt peuvent être effectuées indifféremment dans l'éditeur de méthodes ou le débogueur. Il y a une interaction dynamique entre les deux éditeurs : un point d'arrêt inséré ou modifié dans un éditeur est immédiatement reporté dans l'autre (ainsi que dans la **Liste des points d'arrêt** de l'explorateur d'exécution).

Dans l'exemple suivant, un point d'arrêt a été placé, dans le débogueur, en regard de la ligne **Si ($vrResult>=$vrLimitVal)** :



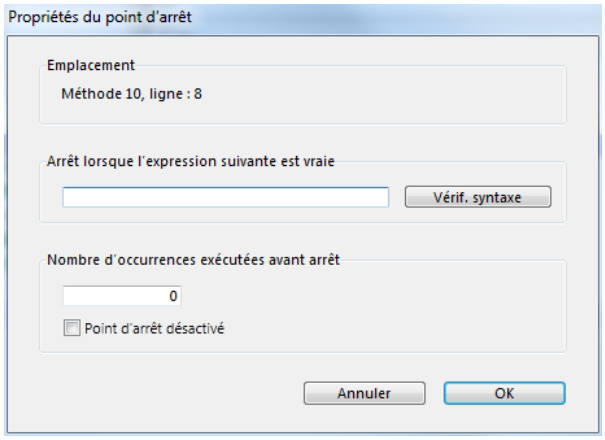
Si vous cliquez de nouveau sur la puce rouge, vous supprimez le point d'arrêt.

Editer un point d'arrêt

Vous pouvez ouvrir la fenêtre **Propriétés du point d'arrêt** en sélectionnant la commande **Modifier point d'arrêt...** dans le menu contextuel de la **Fenêtre d'évaluation des méthodes** ou en utilisant la combinaison **Alt+clic** (sous Windows) / **Option+clic** (sous Mac OS) dans la marge gauche de la fenêtre (ou de l'éditeur de méthodes).

- Si vous avez cliqué sur un point d'arrêt existant, la fenêtre s'ouvre pour ce point d'arrêt.
- Si vous cliquez sur une ligne où aucun point d'arrêt n'a été placé, un point d'arrêt est créé et sa fenêtre de propriétés s'affiche.

Voici la **fenêtre des propriétés du point d'arrêt** :



- **Emplacement** : Vous indique le nom de la méthode et le numéro de la ligne où le point d'arrêt a été placé. Vous ne pouvez pas modifier cette information.
- **Arrêt lorsque l'expression suivante est vraie** : Saisissez une formule 4D qui retourne Vrai ou Faux. Si, par exemple, vous voulez ne vous arrêter à une ligne que lorsque, par exemple, **Enregistrements dans selection([aTable])=0**, saisissez cette formule et l'arrêt se produira uniquement si vous n'avez aucun enregistrement sélectionné pour la table *[aTable]* lorsque le débogueur rencontrera la ligne où le point d'arrêt est placé. Si vous n'êtes pas sûr de la syntaxe de votre formule, cliquez sur le bouton **Vérif. Syntaxe**.
- **Nombre d'occurrences exécutées avant arrêt** : Vous pouvez insérer un point d'arrêt sur une ligne de code placée dans une structure en boucle (Tant que, Repeter, Boucle) ou placée dans une sous-routine ou fonction appelée à partir d'une boucle. Si vous savez que le problème que vous cherchez n'arrivera pas avant au moins la 200e itération de la boucle, saisissez 200 et le point d'arrêt s'activera à la 201e itération.
- **Point d'arrêt désactivé** : Vous avez créé un point d'arrêt dont vous n'avez plus besoin pour le moment mais qui pourrait vous servir plus tard. Il vous suffit dans ce cas de le désactiver. Un point d'arrêt désactivé s'affiche sous forme de tiret (-) et non plus de puce (•) à la fois dans le débogueur, dans l'éditeur de méthodes et dans la **Liste des points d'arrêt**.

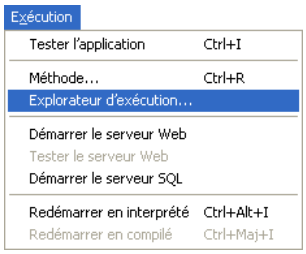
Les points d'arrêt peuvent être créés et édités dans la fenêtre du débogueur et l'éditeur de méthodes. Mais vous pouvez également éditer des points d'arrêt existants en affichant la liste des points d'arrêt, dans la page "Point d'arrêt" de l'Explorateur d'exécution. Pour plus d'informations, reportez-vous à la section **Liste des points d'arrêt**.

Liste des points d’arrêt

La liste des points d’arrêt est une page de l'Explorateur d'exécution qui vous permet de gérer les points d’arrêt que vous avez créés dans la fenêtre du débogueur ou dans l'éditeur de méthodes.

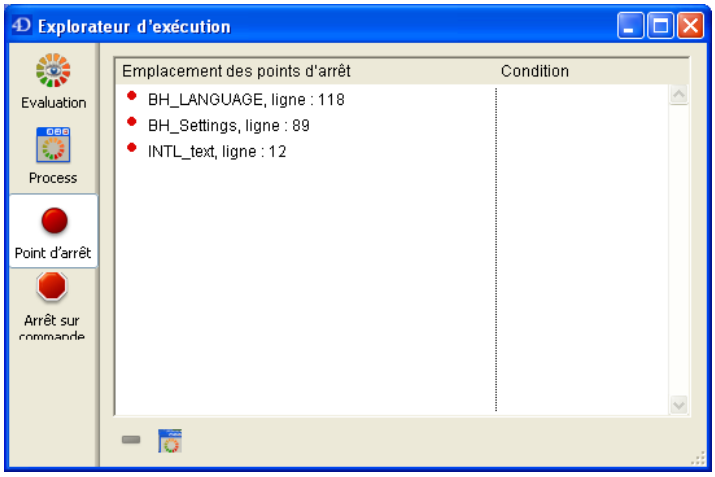
Pour afficher la Liste des points d’arrêt, procédez de la manière suivante :

1. Choisissez **Explorateur d'exécution** dans le menu **Exécution**.
La fenêtre de l'Explorateur d'exécution peut être affichée sous forme de palette flottante (pour cela, maintenez la touche **Maj** enfoncée au moment de sélectionner la commande de menu). L'Explorateur d'exécution est alors disponible dans tous les environnements de 4D. Pour plus d'informations, reportez-vous au manuel Mode Développement.



La fenêtre de l'Explorateur d'exécution apparaît.

2. Cliquez sur le bouton **Points d'arrêt** pour afficher la liste des points d'arrêt :



La liste des points d'arrêts est divisée en deux colonnes :

- La colonne de gauche indique le statut actif/inactif du point d'arrêt, suivi du nom de la méthode et du numéro de la ligne à laquelle le point d'arrêt a été placé dans la fenêtre du débogueur ou de l'éditeur de méthodes.
- La colonne de droite affiche la condition optionnellement associée au point d'arrêt.

Vous pouvez définir dans cette fenêtre une condition pour un point d'arrêt.
Vous pouvez activer, désactiver ou supprimer chaque point d'arrêt. En revanche, vous ne pouvez pas ajouter de point d'arrêt dans l'Explorateur. Les points d'arrêt ne peuvent être créés qu'à partir de la fenêtre du débogueur ou de l'éditeur de méthodes.
Vous pouvez également ouvrir une fenêtre de l'éditeur de méthodes affichant la méthode à laquelle est associé le point d'arrêt en double-cliquant sur le libellé du point d'arrêt.

Définir une condition pour un point d'arrêt

Pour définir une condition pour un point d'arrêt, procédez de la manière suivante :

1. Cliquez dans la colonne de droite en regard du point d'arrêt.
2. Saisissez une formule 4D (expression ou appel de commande ou méthode projet) retournant une valeur booléenne (Vrai ou Faux).

Note : Pour supprimer une condition, effacez la formule correspondante.

Activer/Désactiver un point d'arrêt

Pour activer ou désactiver un point d'arrêt, procédez de la manière suivante :

1. Sélectionnez le point d'arrêt en cliquant dessus ou en utilisant les touches fléchées pour naviguer dans la liste.
2. Sélectionnez la commande **Activer/Désactiver dans le menu contextuel.**

Raccourci : Chaque point d'arrêt dans la liste peut être activé/désactivé par un simple clic sur la puce (•) placée devant son libellé. La puce se transforme en tiret (-) lorsque le point d'arrêt est désactivé.

Supprimer un point d'arrêt

Pour supprimer un point d'arrêt, procédez de la manière suivante :

1. Sélectionnez le libellé du point d'arrêt en cliquant dessus ou en utilisant les touches fléchées pour naviguer dans la liste.
2. Appuyez sur la touche **Retour arrière**, ou cliquez sur le bouton **Supprimer** au-dessous de la liste.

Note : Pour supprimer tous les points d'arrêt, cliquez sur le bouton **Supprimer tout** (deuxième bouton situé sous la liste), ou encore sélectionnez la commande **Supprimer tout** dans le menu contextuel.

Astuces :

- L'ajout de conditions aux points d'arrêt ralentit l'exécution du code car la condition est évaluée à chaque fois que le point d'arrêt est rencontré. D'un autre côté, le débogage est plus rapide car 4D ignore automatiquement les occurrences qui ne satisfont pas aux conditions.
- La désactivation d'un point d'arrêt produit quasiment le même effet que sa suppression. Lors de l'exécution du code, le débogueur ne passe presque pas de temps sur le point d'arrêt. L'avantage de la désactivation est que vous n'avez pas à redéfinir le point d'arrêt en cas de besoin.

Points d'arrêt sur commandes

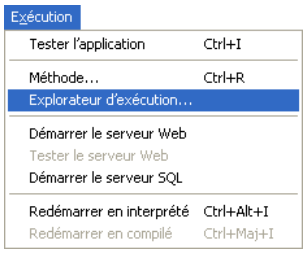
La liste des Points d'arrêt sur commandes est une page de l'Explorateur d'exécution qui vous permet d'ajouter des points d'arrêt supplémentaires dans votre code en interceptant des appels aux commandes 4D.

Placer un point d'arrêt sur une commande vous permet de commencer à tracer l'exécution de n'importe quel process dès qu'une commande particulière est appelée par le process. A la différence d'un point d'arrêt placé dans une méthode projet (qui, par conséquent, déclenche le mode trace uniquement lorsqu'il est atteint), l'aire d'action d'un point d'arrêt sur commande comprend tous les process qui exécutent du code 4D et qui appellent cette commande.

Placer un point d'arrêt sur une commande est un moyen pratique de tracer des grandes parties du code sans devoir insérer des points d'arrêt à des emplacements arbitraires. Si, par exemple, l'exécution dans votre code de plusieurs process durant un certain laps de temps provoque l'effacement d'un enregistrement qui ne devrait théoriquement pas être supprimé, vous pouvez limiter le champ d'investigation en plaçant un point d'arrêt sur les commandes telles que **SUPPRIMER ENREGISTREMENT** et **SUPPRIMER SELECTION**. A chaque fois que ces commandes sont appelées, vous pouvez vérifier si l'enregistrement en question a été supprimé ou non, et donc isoler la partie fautive du code.
Bien entendu, l'expérience aidant, vous pourrez combiner l'utilisation de points d'arrêt dans les méthodes et sur des commandes.

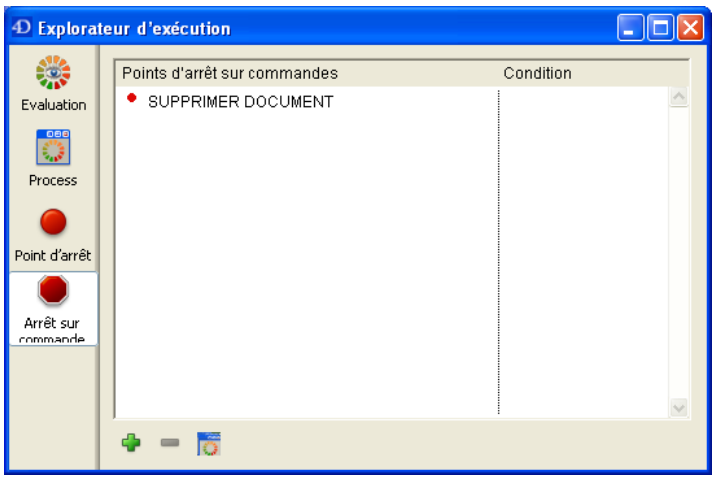
Pour afficher la liste des Points d'arrêt sur commandes, procédez de la manière suivante :

1. Choisissez **Explorateur d'exécution** dans le menu **Exécution**.
La fenêtre de l'Explorateur d'exécution peut être affichée sous forme de palette flottante (pour cela, maintenez la touche **Maj** enfoncée au moment de sélectionner la commande de menu). L'Explorateur d'exécution est alors disponible dans tous les environnements de 4D. Pour plus d'informations, reportez-vous au manuel Mode Développement.



La fenêtre de l'Explorateur d'exécution apparaît.

2. Cliquez sur le bouton **Arrêt sur commande** pour afficher la liste des points d'arrêt sur commande :



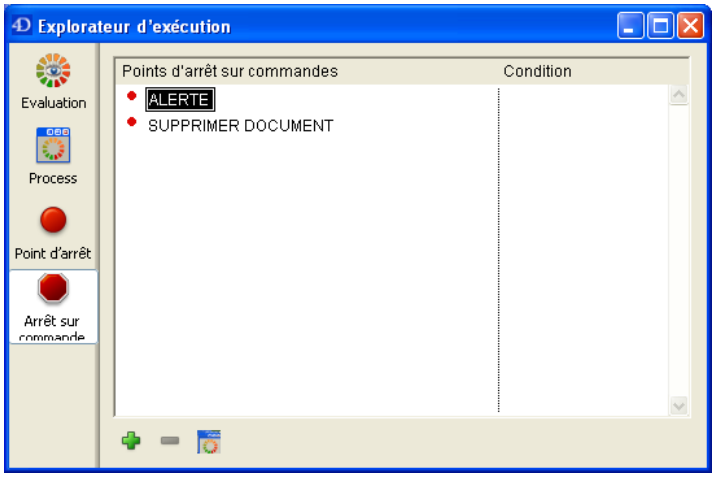
Cette page liste les commandes à intercepter au moment de leur exécution. Elle est divisée en deux colonnes :

- La colonne de gauche indique le statut actif/inactif du point d'arrêt, suivi du nom de la commande.
- La colonne de droite affiche la condition optionnellement associée au point d'arrêt.

Ajouter un nouveau point d'arrêt sur commande

Pour ajouter un nouveau point d'arrêt sur une commande :

1. Cliquez sur le bouton d'ajout **Arrêt sur commande** (en forme de +) situé au-dessous de la liste.
   Une nouvelle entrée est ajoutée dans la liste, avec la commande **ALERTE** par défaut.



Vous pouvez alors cliquer sur le libellé **ALERTE** et saisir le nom de la commande à laquelle vous souhaitez associer un point d'arrêt. Une fois que vous avez terminé, appuyez sur la touche **Entrée** ou **Retour chariot** pour valider votre choix.

Modifier un point d'arrêt sur commande

Pour modifier un point d'arrêt sur une commande, procédez de la manière suivante :

1. Sélectionnez le point d'arrêt en cliquant dessus ou en utilisant les touches fléchées pour naviguer dans la liste (s'il n'y a pas déjà un point d'arrêt sélectionné et en mode édition).
2. Pour passer un point d'arrêt du mode sélection au mode édition et inversement, appuyez sur la touche **Entrée** ou **Retour chariot**.
3. Saisissez ou modifiez le nom de la commande.
4. Pour valider vos modifications, appuyez sur la touche **Entrée** ou **Retour chariot**.

Activer/Désactiver un point d'arrêt sur commande

Pour activer ou désactiver un point d'arrêt sur commande :

1. Cliquez sur la puce (•) placée devant le libellé.
   Cette action permet d'activer ou de désactiver alternativement le point d'arrêt. La couleur de la puce indique le statut courant du point d'arrêt :
   - rouge = activé,
   - orange = désactivé.

Note : La désactivation d'un point d'arrêt sur commande produit quasiment le même effet que sa suppression. Lors de l'exécution du code, le débogueur ne passe presque pas de temps sur le point d'arrêt. L'avantage de la désactivation est que vous n'avez pas à redéfinir le point d'arrêt en cas de besoin.

Supprimer un point d'arrêt sur commande

Pour supprimer un point d'arrêt sur commande, procédez de la manière suivante :

1. Sélectionnez le point d'arrêt en cliquant dessus ou en utilisant les touches fléchées pour naviguer dans la liste (s'il n'y pas déjà un point d'arrêt sélectionné et en mode édition).
2. Appuyez sur la touche **Retour arrière** ou **Suppr** ou cliquez sur le bouton **Supprimer** situé sous la liste.
3. Pour supprimer tous les points d'arrêt, cliquez sur le bouton **Supprimer tout**.

Définir une condition pour un point d'arrêt sur commande

Pour définir une condition pour un point d'arrêt sur commande, procédez de la manière suivante :

1. Cliquez dans la colonne de droite en regard du point d'arrêt.
   Un curseur de saisie apparaît.
2. Saisissez une formule 4D (expression ou appel de commande ou méthode projet) retournant une valeur booléenne (Vrai ou Faux).

Note : Pour supprimer une condition, effacez la formule correspondante.

L'ajout de conditions permet de ne stopper l'exécution lorsque la commande est appelée que si la condition est remplie. Par exemple, si vous associez la condition "**Enregistrements trouves**([Emp]>10)" au point d'arrêt sur la commande **SUPPRIMER SELECTION**, le code ne sera pas stoppé lors de l'exécution de la commande **SUPPRIMER SELECTION** si la sélection courante de la table [Emp] ne contient que 9 enregistrements.

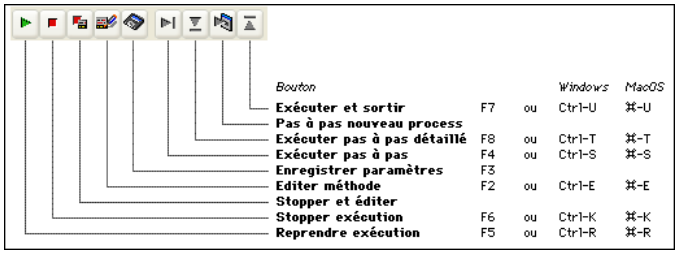
L'ajout de conditions aux points d'arrêt sur commande ralentit l'exécution du code car la condition est évaluée à chaque fois que le point d'arrêt est rencontré. D'un autre côté, le débogage est plus rapide car 4D ignore automatiquement les occurences qui ne satisfont pas aux conditions.

Raccourcis du débogueur

Ce chapitre détaille les raccourcis disponibles dans la fenêtre du débogueur.

Barre d'outils de contrôle d'exécution

L'illustration ci-dessous présente les raccourcis des boutons situés dans la partie supérieure gauche de la fenêtre du débogueur :



La combinaison Maj+F5 ou Maj+clic sur le bouton **Reprendre exécution** reprend l'exécution et en plus désactive tous les appels **TRACE** ultérieurs dans le process courant.

Sous-fenêtre zone d'expression

- un clic avec le bouton droit de la souris (Windows) ou Control+clic (Macintosh) dans la sous-fenêtre d'expression déroule le menu contextuel.
- un double-clic sur un article de la fenêtre d'expression copie cet article dans la fenêtre d'évaluation.

Sous-fenêtre de chaîne d'appel

- Un double-clic sur le nom d'une méthode dans la **Fenêtre de chaîne d'appel** affiche la méthode dans la sous-fenêtre d'évaluation des méthodes, à la ligne correspondant à la chaîne d'appel.

Sous-fenêtre d'évaluation

- Un clic avec le bouton droit de la souris (Windows) ou Control+Clic (Macintosh) dans la **Fenêtre d'évaluation** déroule le menu contextuel de la fenêtre.
- Un double-clic dans la sous-fenêtre d'évaluation crée une nouvelle expression.

Fenêtre d'évaluation des méthodes

- Un clic dans la marge gauche place ou supprime un point d'arrêt.
- Alt+Majuscule+clic (Windows) ou Option+Majuscule+clic (Macintosh) pose un point d'arrêt provisoire
- Alt+clic (Windows) ou Option+clic (Macintosh) affiche la fenêtre des propriétés du point d'arrêt pour un point d'arrêt nouveau ou existant.
- Une expression ou un objet sélectionné(e) peut être copié dans la zone d'évaluation par glisser-déposer.
- Ctrl+D(Windows) ou Commande+D (Mac OS) sur un texte sélectionné le copie dans la zone d'évaluation.

Toutes les sous-fenêtres

- Ctrl+*(Windows) ou Commande+* (Mac OS) force la réactualisation des listes d'expressions.
- Lorsqu'aucun objet n'est sélectionné dans les fenêtres, en appuyant sur **Entrée** vous avancez d'une ligne.
- Lorsque la valeur d'un élément est sélectionnée, utilisez les touches directionnelles pour naviguer dans la liste.
- Lorsque vous êtes en train d'éditer un élément, utilisez les touches directionnelles pour déplacer le curseur, utilisez Ctrl+A/X/C/V (Windows) ou Commande+A/X/C/V (Macintosh) en raccourci des commandes du menu **Edition** : Tout Sélectionner/Couper/Copier/Coller.

4D Write Pro

- WP CREER SIGNET Nouveauté 16.0
- WP EXPORTER DOCUMENT
- WP EXPORTER VARIABLE
- WP FIXER ATTRIBUTS
- WP Importer document
- WP IMPRIMER Modifié 16.0
- WP INSERER DOCUMENT Nouveauté 16.0
- WP INSERER IMAGE Nouveauté 16.0
- WP INSERER RUPTURE Nouveauté 16.0
- WP LIRE ATTRIBUTS
- WP Lire images
- WP Lire paragraphes
- WP Lire plage
- WP Lire plage signet Nouveauté 16.0
- WP Lire selection
- WP LIRE SIGNETS Nouveauté 16.0
- WP Nombre de pages Nouveauté 16.0
- WP Nouveau Modifié 16.0
- WP REINITIALISER ATTRIBUTS
- WP SELECTIONNER
- WP Style pris en charge
- WP SUPPRIMER SIGNET Nouveauté 16.0
- WP UTILISER PARAMETRES IMPRESSION

WP CREER SIGNET

WP CREER SIGNET (objPlage ; nomSignet)

Paramètre	Type		Description
objPlage	Objet	⇒	Plage 4D Write Pro
nomSignet	Chaîne	⇒	Nom du signet à créer

Description

La commande **WP CREER SIGNET** crée un nouveau signet nommé *nomSignet* basé sur la plage 4D Write Pro *objPlage* du document parent.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP EXPORTER DOCUMENT

WP EXPORTER DOCUMENT (docWP ; cheminFichier {; format {; options}})

Paramètre	Type		Description
docWP	Objet	➡	Variable 4D Write Pro
cheminFichier	Chaîne	➡	Chemin du fichier d'export
format	Entier long	➡	Format de sortie du document
options	Entier long	➡	Options d'export

Description

La commande **WP EXPORTER DOCUMENT** exporte l'objet 4D Write Pro *docWP* dans un document sur disque défini par le paramètre *cheminFichier* ainsi que des paramètres optionnels.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP EXPORTER VARIABLE

WP EXPORTER VARIABLE (docWP ; destination ; format {; options})

Paramètre	Type		Description
docWP	Objet	➡	Variable 4D Write Pro
destination	Variable texte, Variable BLOB	⬅	Variable devant recevoir le contenu exporté
format	Entier long	➡	Format de sortie de la variable
options	Entier long, Chaîne	➡	Options d'export

Description

La commande **WP EXPORTER VARIABLE** exporte l'objet 4D Write Pro *docWP* dans la variable 4D *destination* avec le *format* et les éventuelles *options* spécifié(es).

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP FIXER ATTRIBUTS

WP FIXER ATTRIBUTS (objPlage | wpDoc ; nomAttribut ; valeurAttribut {; nomAttribut2 ; valeurAttribut2 ; ... ; nomAttributN ; valeurAttributN})

Paramètre	Type		Description
objPlage | wpDoc	Objet	⇒	Plage ou document 4D Write Pro
nomAttribut	Chaîne	⇒	Nom d'attribut dont vous souhaitez modifier la valeur
valeurAttribut	Chaîne, Réel, Booléen	⇒	Nouvelle valeur de l'attribut

Description

La commande **WP FIXER ATTRIBUTS** vous permet de fixer la valeur d'un ou plusieurs attribut(s) dans une plage ou un document 4D Write Pro.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP Importer document

WP Importer document (cheminFichier) -> Résultat

Paramètre	Type		Description
cheminFichier	Chaîne	→	Chemin du document 4D Write (.4w7 ou .4wt) ou du document 4D Write Pro (.4wp)
Résultat	Objet	←	Objet 4D Write Pro

Description

La commande **WP Importer document** convertit un document 4D Write Pro ou 4D Write existant (extension .4wp, .4w7 ou .4wt) en un nouvel objet 4D Write Pro.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP IMPRIMER

WP IMPRIMER (docWP {; modeImpr})

Paramètre	Type		Description
docWP	Objet	⇨	Nom du document 4D Write Pro
modeImpr	Entier long	⇨	Mode d'impression pour le document 4D Write Pro : 0 (défaut)=Mode 4D Write Pro, 1=Mode HTML WYSIWYG

Description

La commande **WP IMPRIMER** lance une tâche d'impression pour le document 4D Write Pro désigné par *docWP* ou (versions 64 bits uniquement) ajoute le document dans la tâche d'impression courante si elle est appelée entre les commandes **OUVRIR TACHE IMPRESSION** et **FERMER TACHE IMPRESSION**.

Note versions 32 bits : Cette commande est prise en charge dans les versions 32 bits de 4D mais dans ce contexte, elle ne peut pas être appelée à l'intérieur d'une tâche d'impression ouverte avec **OUVRIR TACHE IMPRESSION**.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP INSERER DOCUMENT

WP INSERER DOCUMENT (objPlage ; docWP ; mode {; miseAJourPlage})

Paramètre	Type		Description
objPlage	Objet	⇒	Plage 4D Write Pro
docWP	Objet	⇒	Document 4D Write Pro
mode	Entier long	⇒	Mode d'insertion
miseAJourPlage	Entier long	⇒	Mode de mise à jour de la plage de sélection

Description

La commande **WP INSERER DOCUMENT** insère le document *docWP* dans la plage *objPlage* selon le *mode* d'insertion spécifié et le paramètre *miseAJourPlage*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP INSERER IMAGE

WP INSERER IMAGE (objPlage ; image ; mode {; miseAJourPlage})

Paramètre	Type		Description
objPlage	Objet	⇒	Plage 4D Write Pro
image	Image, Chaîne	⇒	Champ ou variable Image, ou chemin d'accès à une image sur le disque
mode	Entier long	⇒	Mode d'insertion
miseAJourPlage	Entier long	⇒	Mode de mise à jour de la plage de sélection

Description

La commande **WP INSERER IMAGE** insère *image* dans la plage de sélection *objPlage* selon le *mode* d'insertion spécifié.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP INSERER RUPTURE

WP INSERER RUPTURE (objPlage ; typeRupture ; mode {; miseAJourPlage})

Paramètre	Type		Description
objPlage	Objet	⇒	Plage 4D Write Pro
typeRupture	Entier long	⇒	Type de rupture à insérer
mode	Entier long	⇒	Mode d'insertion
miseAJourPlage	Entier long	⇒	Mode de mise à jour de la plage de sélection

Description

La commande **WP INSERER RUPTURE** insère une nouvelle rupture de type *typeRupture* dans la plage de sélection *objPlage* selon le mode d'insertion *mode* et le paramètre *miseAJourPlage*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP LIRE ATTRIBUTS

WP LIRE ATTRIBUTS (objPlage | wpDoc ; nomAttribut ; valeurAttribut {; nomAttribut2 ; valeurAttribut2 ; ... ; nomAttributN ; valeurAttributN})

Paramètre	Type		Description
objPlage | wpDoc	Objet	⇨	Plage ou document 4D Write Pro
nomAttribut	Chaîne	⇨	Nom d'attribut dont vous souhaitez lire la valeur
valeurAttribut	Chaîne, Réel, Booléen, Tableau	⇦	Valeur courante de l'attribut

Description

La commande **WP LIRE ATTRIBUTS** retourne la valeur courante d'attribut(s) dans une plage ou un document 4D Write Pro.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP Lire images

WP Lire images (objPlage) -> Résultat

Paramètre	Type		Description
objPlage	Objet	⇨	Plage dans laquelle lire les images
Résultat	Objet	⇦	Plage contenant uniquement les images

Description

La commande **WP Lire images** retourne un objet plage spécifique qui référence uniquement les images contenues dans *objPlage* que vous avez passé en paramètre.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP Lire paragraphes

WP Lire paragraphes (objPlage) -> Résultat

Paramètre	Type		Description
objPlage	Objet	⇨	Plage dans laquelle lire les paragraphes
Résultat	Objet	⇦	Plage définissant uniquement les paragraphes

Description

La commande **WP Lire paragraphes** retourne un objet plage spécifique qui référence uniquement les paragraphes contenus dans *objPlage* que vous avez passé en paramètre.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP Lire plage

WP Lire plage (zoneWP ; débutPlage ; finPlage) -> Résultat

Paramètre	Type		Description
zoneWP	Objet	⇒	Champ ou variable objet 4D Write Pro
débutPlage	Entier long	⇒	Position du début de la plage dans la zone
finPlage	Entier long	⇒	Position de la fin de la plage dans la zone
Résultat	Objet	⇐	Nouvel objet plage

Description

La commande **WP Lire plage** retourne une nouvelle plage de sélection (*objPlage*) contenant les caractères situés entre *débutPlage* et *finPlage* dans la zone 4D Write Pro *zoneWP*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP Lire plage signet

WP Lire plage signet (docWP ; nomSignet) -> Résultat

Paramètre	Type		Description
docWP	Objet	⇒	Document 4D Write Pro
nomSignet	Texte	⇒	Nom du signet dont vous souhaitez récupérer la plage
Résultat	Objet	⇐	Plage du signet

Description

La commande **WP Lire plage signet** retourne un objet de type plage (*objPlage*) contenant la plage associée au signet nommé *nomSignet* dans le document *docWP*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP Lire selection

WP Lire selection ({* ;} zoneWP) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, zoneWP est un nom d'objet (chaîne). Si omis, zoneWP est un champ ou une variable
zoneWP	Chaîne, Objet	⇒	Nom d'objet de formulaire (si * spécifié) ou champ ou variable objet 4D Write Pro (si * omis)
Résultat	Objet	⇐	Nouvel objet plage

Description

La commande **WP Lire selection** retourne une nouvelle plage de sélection (*objPlage*) basée sur la sélection courante de texte dans la zone 4D Write Pro *zoneWP*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP LIRE SIGNETS

WP LIRE SIGNETS (docWP ; tabNomsSignets)

Paramètre	Type		Description
docWP	Objet	➡	Document 4D Write Pro
tabNomsSignets	Tableau texte	⬅	Tableau des noms de signets

Description

La commande **WP LIRE SIGNETS** retourne un tableau contenant le nom de tous les signets définis dans le document *docWP*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP Nombre de pages

WP Nombre de pages (docWP) -> Résultat

Paramètre	Type		Description
docWP	Objet	⇒	Document 4D Write Pro
Résultat	Entier long	⇐	Nombre de pages du document

Description

La commande **WP Nombre de pages** retourne le nombre de pages courant du document 4D Write Pro *docWP*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP Nouveau

WP Nouveau {(source)} -> Résultat

Paramètre	Type		Description
source	Chaîne, BLOB, Objet	⇒	Chaîne : Source HTML 4D, BLOB : Blob document 4D Write (.4w7/.4wt) ou document 4D Write Pro (.4wp), Objet : Objet plage 4D Write Pro
Résultat	Objet	⇐	Objet 4D Write Pro

Description

La commande **WP Nouveau** crée et retourne un nouvel objet 4D Write Pro vide ou avec le contenu défini par *source*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP REINITIALISER ATTRIBUTS

WP REINITIALISER ATTRIBUTS (objPlage ; nomAttribut {; nomAttribut2 ; ... ; nomAttributN})

Paramètre	Type		Description
objPlage	Objet	⇒	Plage 4D Write Pro
nomAttribut	Chaîne	⇒	Nom(s) d'attribut(s) à réinitialiser

Description

La commande **WP REINITIALISER ATTRIBUTS** vous permet de réinitialiser la valeur d'un ou plusieurs attributs dans la plage 4D Write Pro *objPlage*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP SELECTIONNER

WP SELECTIONNER ({* ;} zoneWP {; objPlage}{; débutPlage ; finPlage})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, zoneWP est un nom d'objet (chaîne). Si omis, zoneWP est un champ ou une variable
zoneWP	Chaîne, Objet	⇒	Nom d'objet de formulaire (si * spécifié) ou champ ou variable objet 4D Write Pro (si * omis)
objPlage	Objet	⇒	Objet plage à utiliser pour créer une sélection
débutPlage	Entier long	⇒	Position du début de la plage
finPlage	Entier long	⇒	Position de la fin de la plage

Description

La commande **WP SELECTIONNER** crée une nouvelle sélection de texte dans la zone *zoneWP* de 4D Write Pro, basée sur la plage *objPlage* ou la nouvelle plage définie par *débutPlage* et *finPlage*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP Style pris en charge

WP Style pris en charge (objRange ; stylePoliceWP) -> Résultat

Paramètre	Type		Description
objRange	Objet	⇒	Plage à analyser
stylePoliceWP	Entier long	⇒	Constante de style de police : wk font bold, wk font italic, wk text underline style, wk text linethrough style
Résultat	Booléen	⇐	Vrai si le style est pris en charge dans tout ou partie de la plage, Faux sinon

Description

La commande **WP Style pris en charge** retourne Vrai si le style *stylePoliceWP* est pris en charge par au moins une partie du texte inclus dans *objPlage*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP SUPPRIMER SIGNET

WP SUPPRIMER SIGNET (docWP ; nomSignet)

Paramètre	Type		Description
docWP	Objet	⇨	Document 4D Write Pro
nomSignet	Chaîne	⇨	Nom du signet à supprimer

Description

La commande **WP SUPPRIMER SIGNET** supprime de *docWP* le signet nommé *nomSignet*.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

WP UTILISER PARAMETRES IMPRESSION

WP UTILISER PARAMETRES IMPRESSION (docWP)

Paramètre	Type		Description
docWP	Objet	⇨	Nom du document 4D Write Pro

Description

La commande **WP UTILISER PARAMETRES IMPRESSION** modifie les options d'impression de la page courante, sur la base des attributs du document 4D Write Pro *docWP* pour la taille de page et l'orientation.

Pour plus d'informations, reportez-vous à la description de cette commande dans le chapitre **4D Write Pro - Langage** du *Guide de référence* de 4D Write Pro.

Accès objets développement

- Commandes du thème Accès objets développement
- Chemin methode courante
- FORM LIRE NOMS
- METHODE FIXER ATTRIBUT
- METHODE FIXER ATTRIBUTS
- METHODE FIXER CODE
- METHODE FIXER COMMENTAIRES
- METHODE FIXER MODE ACCES
- METHODE Lire attribut
- METHODE LIRE ATTRIBUTS
- METHODE Lire chemin
- METHODE LIRE CHEMINS
- METHODE LIRE CHEMINS FORM
- METHODE LIRE CODE
- METHODE LIRE COMMENTAIRES
- METHODE LIRE DATE MODIFICATION
- METHODE LIRE DOSSIERS
- METHODE LIRE NOMS
- METHODE OUVRIR CHEMIN
- METHODE RESOUDRE CHEMIN

Commandes du thème Accès objets développement

4D vous permet d'accéder par programmation au contenu des méthodes de vos applications. Ce *source toolkit* facilite l'intégration de vos applications aux outils de contrôle du code, notamment les applications de gestion de versions (VCS). Il permet également de mettre en place des systèmes avancés de documentation du code, de construire un explorateur personnalisé ou encore d'organiser la sauvegarde régulière du code sous forme de fichiers sur disque.

Les principes suivants sont mis en oeuvre :

- Chaque méthode et formulaire d'une application 4D dispose d'une adresse sous forme de chemin d'accès. Par exemple, la méthode trigger de la table 1 est accessible à l'adresse "[trigger]/table_1". Chaque chemin d'accès d'objet est unique dans une application.
  Note : Pour assurer l'unicité des chemins d'accès, 4D ne permet plus de créer des objets de même nom dans des pages formulaires différentes. Dans les bases de données converties depuis des versions antérieures à 4D v13, le CSM vous permet de détecter ces doublons.
- L'accès aux objets de l'application 4D s'effectue à l'aide des commandes de ce thème, par exemple **METHODE LIRE NOMS** ou **METHODE LIRE CHEMINS**.
- La majorité des commandes de ce thème fonctionnent en mode interprété et en mode compilé. Les commandes modifiant les propriétés ou accédant au contenu exécutable des méthodes peuvent être utilisées en mode interprété uniquement (cf. tableau ci-dessous).
- Toutes les commandes de ce thème sont utilisables avec 4D en mode local ou distant.
  En revanche, gardez à l'esprit que certaines commandes ne sont pas utilisables en mode compilé : la finalité du thème est la création d'outils personnalisés d'aide de développement. Les commandes ne doivent pas être utilisées pour modifier dynamiquement le fonctionnement d'une base en exécution. Par exemple, vous ne pouvez pas utiliser **METHODE FIXER ATTRIBUT** pour modifier un attribut de méthode en fonction du statut de l'utilisateur courant.
- Lorsqu'une commande de ce thème est appelée depuis un composant, elle accède par défaut aux objets du composant. Pour accéder aux objets de la base hôte dans ce cas, il suffit de passer un * en dernier paramètre. A noter que cette syntaxe est la seule possible dans ce contexte pour les commandes permettant de modifier les objets (telles que **METHODE FIXER ATTRIBUT**), car les composants sont toujours exécutés en lecture seulement.

Utilisation en mode compilé

Pour des raisons liées au principe même du processus de compilation, seules certaines commandes de ce thème sont utilisables en mode compilé. Le tableau suivant indique la disponibilité des commandes en mode compilé :

Commande	Utilisable en mode compilé
Chemin methode courante	Oui
FORM LIRE NOMS	Oui
METHODE FIXER ATTRIBUT	Non (*)
METHODE FIXER ATTRIBUTS	Non (*)
METHODE FIXER CODE	Non (*)
METHODE FIXER COMMENTAIRES	Non (*)
METHODE FIXER MODE ACCES	Oui
METHODE Lire attribut	Oui
METHODE LIRE ATTRIBUTS	Oui
METHODE Lire chemin	Oui
METHODE LIRE CHEMINS	Oui
METHODE LIRE CHEMINS FORM	Oui
METHODE LIRE CODE	Non (*)
METHODE LIRE COMMENTAIRES	Oui
METHODE LIRE DATE MODIFICATION	Oui
METHODE LIRE DOSSIERS	Oui
METHODE LIRE NOMS	Oui
METHODE OUVRIR CHEMIN	Non (*)
METHODE RESOUDRE CHEMIN	Oui

(*) L'erreur -9762, "La commande ne peut pas être exécutée dans une base compilée." est générée lorsque la commande est exécutée en mode compilé.

Construction des chemins d'accès

Par défaut, aucun fichier n'est créé sur disque par 4D. Cependant, les chemins d'accès générés pour les objets sont compatibles avec la gestion de fichiers du système d'exploitation, ils peuvent être utilisés directement pour générer des fichiers sur disque via vos propres méthodes d'import/export.

En particulier, les caractères interdits tels que ":" sont encodés dans les noms des méthodes. Les fichiers générés pourront être automatiquement intégrés à une application de gestion de versions. Les caractères encodés sont les suivants :

Caractère	Encodage
"	%22
*	%2A
/	%2F
:	%3A
<	%3C
>	%3E
?	%3F
|	%7C
\	%5C
%	%25

Exemples :
Form?1 sera encodé *Form%3F1*
Button/1 sera encodé *Button%2F1*

Chemin methode courante

Chemin methode courante -> Résultat

Paramètre	Type		Description
Résultat	Texte	➲	Chemin interne complet de la méthode en cours d'exécution

Description

La commande **Chemin methode courante** retourne le chemin d'accès interne de la méthode base, du trigger, de la méthode projet, méthode formulaire ou méthode objet en cours d'exécution.

Note : Dans le contexte des macro-commandes 4D, la balise *<method_path>* est remplacée par le chemin d'accès complet du code en cours d'édition.

FORM LIRE NOMS

FORM LIRE NOMS ({laTable ;} tabNoms {; filtre {; marqueur}}{; *})

Paramètre	Type		Description
laTable	Table	⇒	Référence de table
tabNoms	Tableau texte	⇐	Tableau des noms de formulaires
filtre	Texte	⇒	Filtrage des noms
marqueur	Entier long	⇒	Marqueur de version minimale à retourner
		⇐	Nouvelle valeur
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **FORM LIRE NOMS** remplit le tableau *tabNoms* avec les noms des formulaires de l'application.

Si vous passez le paramètre *laTable*, la commande retourne les noms des formulaires table associés à cette table. Si vous omettez ce paramètre, le commande retourne les noms des formulaires projet de la base.

Vous pouvez restreindre la liste des formulaires en passant une chaîne de comparaison dans le paramètre *filtre* : dans ce cas, seuls les formulaires dont le nom correspond au filtre seront retournés. Vous pouvez utiliser le caractère @ afin de définir des filtres de type "commence par", "se termine par" ou "contient". Si vous passez une chaîne vide, le paramètre *filtre* est ignoré.

Vous pouvez également restreindre la liste des formulaires à l'aide du paramètre optionnel *marqueur*. Ce paramètre permet de limiter les formulaires retournés dans *tabNoms* à ceux qui ont été modifiés ultérieurement à un instant donné. Dans le cadre d'un système de contrôle de version, ce paramètre vous permet de ne mettre à jour que les formulaires ayant été modifiés depuis la dernière sauvegarde.
Ce principe fonctionne de la manière suivante : 4D maintient en interne un compteur de modification des ressources de la base. Les formulaires étant des ressources, chaque fois qu'un formulaire est créé ou réenregistré, ce compteur est incrémenté. Si vous passez le paramètre *marqueur*, la commande retourne dans *tabNoms* uniquement les formulaires correspondant à des valeurs de marqueurs supérieures ou égales à celle de *marqueur*. En outre, si vous passez une variable dans *marqueur*, la commande retourne dans cette variable la nouvelle valeur du compteur de modification, c'est-à-dire la plus élevée. Vous pouvez alors sauvegarder cette valeur et l'utiliser lors du prochain appel de la commande **FORM LIRE NOMS** afin de ne récupérer que les formulaires nouveaux ou modifiés.

Si la commande est exécutée depuis un composant, elle retourne par défaut les noms des formulaires projet du composant. Si vous passez le paramètre *, le tableau contiendra les formulaires de la base hôte.

Note : Les formulaires placés dans la corbeille ne sont pas listés.

Exemple

Exemples d'utilisations type :

```
 // Liste de tous les formulaires projet de la base
FORM LIRE NOMS(t_Noms)

    // Liste des formulaires de la table [Emps]
FORM LIRE NOMS([Emps];t_Noms)

    // Liste des formulaires "input" de la table [Emps]
FORM LIRE NOMS([Emps];t_Noms;"input_@")

    // Liste de formulaires projet spécifiques de la base
FORM LIRE NOMS(t_Noms;"dialogue_@")

 // Liste de tous les formulaires projet de la base modifiés depuis la dernière synchronisation
 // vMarqueur contient une valeur numérique
FORM LIRE NOMS(t_Noms;"";vMarqueur)

    // Liste de formulaires table depuis un composant
    // Un pointeur est requis car le nom de la table est inconnu
FORM LIRE NOMS(tablePtr->;t_Noms;*)
```

METHODE FIXER ATTRIBUT

METHODE FIXER ATTRIBUT (chemin ; typeAttribut ; valeurAttribut {; typeAttribut2 ; valeurAttribut2 ; ... ; typeAttributN ; valeurAttributN}{; *})

Paramètre	Type		Description
chemin	Texte	⇒	Chemin de méthode projet
typeAttribut	Entier long	⇒	Type d'attribut
valeurAttribut	Booléen, Texte	⇒	Vrai = sélectionner l'attribut, Faux = désélectionner l'attribut ou Nom du dossier
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE FIXER ATTRIBUT** permet de définir la valeur d'un ou plusieurs attribut(s) *typeAttribut* pour la méthode projet désignée par le paramètre *chemin*. Cette commande ne fonctionne qu'avec les méthodes projet. Si vous passez un *chemin* invalide, une erreur est générée.

Passez dans le paramètre *typeAttribut* une valeur indiquant le type d'attribut à définir. Vous pouvez utiliser les constantes suivantes, placées dans le thème **Accès objets développement** :

Constante	Type	Valeur	Comment
Attribut exécutée sur serveur	Entier long	8	Correspond à l'option "Exécuter sur serveur"
Attribut invisible	Entier long	1	Correspond à l'option "Invisible"
Attribut nom dossier	Entier long	1024	Nom de dossier pour la méthode (attribut "folder"). Lorsque vous passez cette constante, vous devez passer un nom de dossier dans *valeurAttribut* : • si le nom correspond à un dossier valide, la méthode sera placée dans ce dossier parent, • si le dossier n'existe pas, la commande ne change rien au niveau du dossier parent, • si vous passez une chaîne vide, la méthode sera placée au niveau racine.
Attribut partagée	Entier long	5	Correspond à l'option "Partagée entre composants et base hôte"
Attribut publiée SOAP	Entier long	3	Correspond à l'option "Offerte comme Web Service"
Attribut publiée SQL	Entier long	7	Correspond à l'option "Disponible via SQL"
Attribut publiée Web	Entier long	2	Correspond à l'option "Disponible via les balises HTML et les URLs 4D (4DACTION...)"
Attribut publiée WSDL	Entier long	4	Correspond à l'option "Publiée dans WSDL". N'est prise en compte que si l'option "Offerte comme Web Service" est activée.

Passez dans le paramètre *valeurAttribut* soit :

- **Vrai** pour sélectionner l'option correspondante et **Faux** pour la désélectionner,
- une chaîne (nom du dossier) si vous avez utilisé la constante Attribut nom dossier dans *typeAttribut*.

Vous pouvez passer plusieurs paires *typeAttribut*;*valeurAttribut* en un seul appel.

Vous pouvez exécuter cette commande depuis un composant, mais dans ce cas vous devez passer le paramètre * car l'accès en écriture au code du composant n'est pas possible. Si vous omettez le paramètre * dans ce contexte, l'erreur -9763 est générée.

Cette commande ne peut pas être exécutée en mode compilé. Dans ce mode, son appel génère l'erreur -9762.

Exemple 1

Sélection de la propriété "Partagée entre composants et base hôte" pour la méthode projet "Choix dialogue" :

```
METHODE FIXER ATTRIBUT("Choix dialogue";Attribut partagée;Vrai)
```

Exemple 2

Définition de plusieurs paires attribut/valeur :

```
METHODE FIXER ATTRIBUT(vChemin;Attribut invisible;vInvisible;Attribut publiée Web;v4DAction;Attribut publiée
SOAP;vSoap;Attribut publiée WSDL;vWSDL;Attribut partagée;vExported;Attribut publiée SQL;vSQL;Attribut exécutée sur
serveur;vRemote;Attribut nom dossier;vDossier;*)
```

METHODE FIXER ATTRIBUTS

METHODE FIXER ATTRIBUTS (chemin ; attributs {; *})

Paramètre	Type		Description
chemin	Texte, Tableau texte	⇒	Chemin(s) de méthode(s)
attributs	Objet, Tableau objet	⇒	Attribut(s) de méthode(s) à définir
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE FIXER ATTRIBUTS** vous permet de définir les valeurs des *attributs* pour la ou les méthode(s) spécifiée(s) dans le paramètre *chemin*.

Dans le paramètre *chemin*, vous pouvez passer soit un texte contenant un chemin de méthode, soit un tableau texte contenant un tableau de chemins. Vous devrez passer le même type de paramètre (variable simple ou tableau) dans le paramètre *attributs* afin de définir les valeurs adéquates. Cette commande ne fonctionne qu'avec les méthodes projet. Si vous passez un *chemin* invalide, une erreur est générée.

Dans le paramètre *attributs*, vous pouvez passer un objet ou un tableau d'objets, selon le type de paramètre passé dans *chemin*, contenant tous les attributs à fixer pour la ou les méthode(s).

Les attributs de méthodes doivent être définis à l'aide des commandes **OB FIXER** ou **OB FIXER TABLEAU**, avec les valeurs Vrai or Faux pour les attributs booléens, ou des valeurs spécifiques pour les attributs étendus (par exemple, "scope":"table" pour la propriété 4D Mobile). Seuls les attributs présents dans le paramètre *attributs* seront mis à jour dans les attributs des méthodes.

Si la commande est exécutée depuis un composant, elle s'applique par défaut aux méthodes du composant. Si vous passez le paramètre *, elle accède aux méthodes de la base hôte.

Note : La commande existante **METHODE FIXER ATTRIBUT** reste prise en charge, toutefois comme elle ne peut retourner que des valeurs booléennes, elle ne peut pas être utilisée pour les attributs étendus tels que les propriétés 4D Mobile.

Les attributs pris en charge sont :

```
{     "invisible" : false, // true si visible     "preemptive" : "capable" // ou bien "incapable" ou "indifferent"     "publishedWeb"
: false,  // true si disponible via les balised et URLs 4D     "publishedSoap": false,  // true si offerte comme Web Service
"publishedWsdl": false,  // true si publiée dans WSDL     "shared" : false,  // true si partagée entre composants et base hôte
"publishedSql" : false,  // true si disponible via SQL     "executedOnServer" : false, // true si exécutée sur le serveur
"published4DMobile" : {         "scope": "table",  // "none" ou "table" ou "currentRecord" ou "currentSelection"         "table":
"nomTable"  // présent si scope est différent de "none"     } }
```

Note : Pour les attributs "published4DMobile", si la valeur "table" n'existe pas ou si le "scope" est invalide, ces attributs sont ignorés.

Exemple 1

Vous souhaiter modifier un seul attribut :

```
C_OBJET($attributes)
OB FIXER($attributes;"executedOnServer";Vrai)
METHODE FIXER ATTRIBUTS("aMethod";$attributes) //seul l'attribut "executedOnServer" est modifié
```

Exemple 2

Vous souhaitez qu'une méthode soit indisponible pour 4D Mobile (la valeur "none" doit être passée à l'attribut "scope") :

```
C_OBJET($attributes)
C_OBJET($fourDMobileAttribute)
OB FIXER($fourDMobileAttribute;"scope";"none")
OB FIXER($attributes;"published4DMobile";$fourDMobileAttribute)
METHODE FIXER ATTRIBUTS("aMethod";$attributes)
```

METHODE FIXER CODE

METHODE FIXER CODE (chemin ; code {; *})

Paramètre	Type		Description
chemin	Texte, Tableau texte	⇒	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
code	Texte, Tableau texte	⇒	Code de la ou des méthode(s) désignée(s)
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE FIXER CODE** modifie le code de la ou des méthode(s) désignée(s) par le paramètre *chemin* avec le contenu passé dans le paramètre *code*. La commande peut accéder au code de tous les types de méthodes : méthodes base, triggers, méthodes projet, méthodes formulaire et méthodes objet.

Dans le cas d'une méthode projet, si la méthode existe déjà dans la base, son contenu est remplacé ; si elle n'existe pas déjà, elle est créée avec son contenu.

Vous pouvez utiliser deux types de syntaxes, basées soit sur des tableaux texte, soit sur des variables texte :

```
C_TEXTE(vTchemin) // variables texte
C_TEXTE(vTcode)
METHODE FIXER CODE(vTchemin;vTcode) // code d'une seule méthode
```

```
TABLEAU TEXTE(tabChemins;0) // tableaux texte
TABLEAU TEXTE(tabCodes;0)
METHODE FIXER CODE(tabChemins;tabCodes) // codes de plusieurs méthodes
```

Il n'est pas possible de mixer les deux syntaxes.

Si un chemin d'accès passé est invalide, la commande ne fait rien.

Lors de l'appel de **METHODE FIXER CODE**, par défaut les attributs des méthodes sont réinitialisés. Cependant, si la première ligne du *code* d'une méthode contient des métadonnées valides (exprimées en JSON), elles sont utilisées pour définir les attributs de la méthode et la première ligne n'est pas insérée. Exemple de métadonnées :

```
// %attributes = {"invisible":true,"lang":"fr","folder":"Security"}
```

Note : Ces métadonnées sont générées automatiquement par la commande **METHODE LIRE CODE**. Pour plus d'informations sur les attributs pris en charge, reportez-vous à la description de la commande **METHODE FIXER ATTRIBUTS**.

Concernant la propriété "folder" des métadonnées :

- si cette propriété est présente et correspond à un dossier valide, la méthode sera placée dans le dossier parent,
- si cette propriété n'est pas présente ou si le dossier n'existe pas, la commande ne change rien au niveau du dossier parent,
- si cette propriété est présente et contient une chaîne vide, la méthode sera placée au niveau racine.

Vous pouvez exécuter cette commande depuis un composant, mais dans ce cas vous devez passer le paramètre * car l'accès en écriture au code du composant n'est pas possible. Si vous omettez le paramètre * dans ce contexte, l'erreur -9763 est générée.

Exemple

Cet exemple permet d'exporter et d'importer la totalité des méthodes projet d'une application :

```
$root_t:=Dossier 4D(Dossier base)+"methods"+Séparateur dossier
TABLEAU TEXTE($fileNames_at;0)
CONFIRMER("Import ou export des méthodes ?";"Import";"Export")

Si(OK=1)
   LISTE DES DOCUMENTS($root_t;$fileNames_at)
   Boucle($loop_l;1;Taille tableau($fileNames_at))
      $filename_t:=$fileNames_at{$loop_l}
      DOCUMENT VERS BLOB($root_t+$filename_t;$blob_x)
      METHODE FIXER CODE($filename_t;BLOB vers texte($blob_x;UTF8 texte sans longueur))
   Fin de boucle
Sinon
   Si(Tester chemin acces($root_t)#Est un dossier)
      CREER DOSSIER($root_t;*)
   Fin de si
   METHODE LIRE CHEMINS(Chemin Méthode projet;$fileNames_at)
   METHODE LIRE CODE($fileNames_at;$code_at)
   Boucle($loop_l;1;Taille tableau($fileNames_at))
      $filename_t:=$fileNames_at{$loop_l}
      FIXER TAILLE BLOB($blob_x;0)
      TEXTE VERS BLOB($code_at{$loop_l};$blob_x;UTF8 texte sans longueur)
      BLOB VERS DOCUMENT($root_t+$filename_t;$blob_x)
   Fin de boucle
```

```
    Fin de si
    MONTRER SUR DISQUE($root_t)
```

METHODE FIXER COMMENTAIRES

METHODE FIXER COMMENTAIRES (chemin ; commentaires {; *})

Paramètre	Type		Description
chemin	Texte, Tableau texte	⇒	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
commentaires	Texte, Tableau texte	⇒	Commentaires de(s) méthode(s) désignée(s)
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE FIXER COMMENTAIRES** remplace les commentaires de la ou des méthode(s) désignée(s) par le paramètre *chemin* par ceux définis dans le paramètre *commentaires*.

Les commentaires modifiés par cette commande sont ceux définis dans l'Explorateur de 4D (à ne pas confondre avec les lignes de commentaires dans le code). Ces commentaires peuvent être générés uniquement pour les méthodes de type triggers, méthodes projet et méthodes formulaire. Ils contiennent du texte stylé.

Note : Les formulaires et les méthodes formulaire partagent les mêmes commentaires.

Vous pouvez utiliser deux types de syntaxes, basées soit sur des tableaux texte, soit sur des variables texte :

```
C_TEXTE(vTchemin) // variables texte
C_TEXTE(vTcommentaires)
METHODE FIXER COMMENTAIRES(vTchemin;vTcommentaires) // commentaires d'une seule méthode
```

```
TABLEAU TEXTE(tabChemins;0) // tableaux texte
TABLEAU TEXTE(tabCommentaires;0)
METHODE FIXER COMMENTAIRES(tabChemins;tabCommentaires) // commentaires de plusieurs méthodes
```

Il n'est pas possible de mixer les deux syntaxes.

Si un chemin d'accès passé est invalide, une erreur est générée.

Vous pouvez exécuter cette commande depuis un composant, mais dans ce cas vous devez passer le paramètre * car l'accès en écriture au code du composant n'est pas possible. Si vous omettez le paramètre * dans ce contexte, l'erreur -9763 est générée.

Exemple

Ajout d'une date de modification à un commentaire de trigger existant :

```
METHODE LIRE COMMENTAIRES("[trigger]/Table1";$comments)
$comments:="Modif :"+Chaine(Date du jour)+"\r"+$comments
METHODE FIXER COMMENTAIRES("[trigger]/Table1";$comments)
```

METHODE FIXER MODE ACCES

METHODE FIXER MODE ACCES (mode)

Paramètre	Type		Description
mode	Entier long	⇨	Mode d'accès aux objets verrouillés

Description

La commande **METHODE FIXER MODE ACCES** vous permet de définir le comportement de 4D lorsque vous tentez d'accéder en écriture à un objet déjà chargé en modification par un autre utilisateur ou process.La portée de cette commande est la session.

Passez dans *mode* une des constantes suivantes du thème **Accès objets développement** :

Constante	Type	Valeur	Comment
Sur objet verrouillé abandonner	Entier long	0	Le chargement de l'objet est abandonné (fonctionnement par défaut)
Sur objet verrouillé confirmer	Entier long	2	4D affiche une boîte de dialogue vous permettant de choisir de réessayer ou d'abandonner. En mode distant, cette option n'est pas prise en charge (le chargement est abandonné)
Sur objet verrouillé réessayer	Entier long	1	4D tente de charger l'objet jusqu'à ce qu'il soit libéré

METHODE Lire attribut

METHODE Lire attribut (chemin ; typeAttribut {; *}) -> Résultat

Paramètre	Type		Description
chemin	Texte	⇒	Chemin de méthode projet
typeAttribut	Entier long	⇒	Type d'attribut à obtenir
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)
Résultat	Booléen	⇐	Vrai = attribut sélectionné, sinon Faux

Description

La commande **METHODE Lire attribut** retourne la valeur de l'attribut *typeAttribut* pour la méthode projet désignée par le paramètre *chemin*. Cette commande ne fonctionne qu'avec les méthodes projet. Si vous passez un *chemin* invalide, une erreur est générée.

Passez dans le paramètre *typeAttribut* une valeur indiquant le type d'attribut à lire. Vous pouvez utiliser les constantes suivantes, placées dans le thème **Accès objets développement** :

Constante	Type	Valeur	Comment
Attribut exécutée sur serveur	Entier long	8	Correspond à l'option "Exécuter sur serveur"
Attribut invisible	Entier long	1	Correspond à l'option "Invisible"
Attribut partagée	Entier long	5	Correspond à l'option "Partagée entre composants et base hôte"
Attribut publiée SOAP	Entier long	3	Correspond à l'option "Offerte comme Web Service"
Attribut publiée SQL	Entier long	7	Correspond à l'option "Disponible via SQL"
Attribut publiée Web	Entier long	2	Correspond à l'option "Disponible via les balises HTML et les URLs 4D (4DACTION...)"
Attribut publiée WSDL	Entier long	4	Correspond à l'option "Publiée dans WSDL". N'est prise en compte que si l'option "Offerte comme Web Service" est activée.

Si la commande est exécutée depuis un composant, elle s'applique par défaut aux méthodes du composant. Si vous passez le paramètre *, elle accède aux méthodes de la base hôte.

La commande retourne **Vrai** si un attribut est sélectionné et **Faux** s'il est désélectionné.

METHODE LIRE ATTRIBUTS

METHODE LIRE ATTRIBUTS (chemin ; attributs {; *})

Paramètre	Type		Description
chemin	Texte, Tableau texte	⇒	Chemin(s) de méthode(s)
attributs	Objet, Tableau objet	⇐	Attribut(s) de méthode(s)
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE LIRE ATTRIBUTS** retourne, dans le paramètre *attributs*, la valeur courante de tous les attributs de la ou des méthode(s) spécifiée(s) dans le paramètre *chemin*.

Cette commande ne fonctionne qu'avec les méthodes projet. Si vous passez un *chemin* invalide, une erreur est générée.

Dans le paramètre *chemin*, vous pouvez passer soit un texte contenant un chemin de méthode, soit un tableau texte contenant un tableau de chemins. Vous devrez passer le même type de paramètre (variable simple ou tableau) dans le paramètre *attributs* afin de récupérer les valeurs adéquates.

Dans le paramètre *attributs*, vous pouvez passer un objet ou un tableau d'objets, selon le type de paramètre passé dans *chemin*. Tous les attributs de méthode(s) sont retournés sous forme de propriétés d'objet, avec des valeurs "True"/"False" pour les attributs Booléens, des valeurs texte ou des valeurs supplémentaires si nécessaire (par exemple, "scope":"table" pour la propriété 4D Mobile).

Si la commande est exécutée depuis un composant, elle s'applique par défaut aux méthodes du composant. Si vous passez le paramètre *, elle accède aux méthodes de la base hôte.

Note : La commande existante **METHODE Lire attribut** reste prise en charge, toutefois comme elle ne peut retourner que des valeurs booléennes, elle ne peut pas être utilisée pour les attributs étendus tels que les propriétés 4D Mobile.

Exemple

Vous souhaitez connaître les attributs de la méthode projet *sendMail*. Vous pouvez écrire :

```
C_OBJET($att)
METHODE LIRE ATTRIBUTS("sendMail";$att)
```

A l'issue de l'exécution, $att contient, par exemple :

```
{     "invisible":false,     "preemptive":"capable",     "publishedWeb":false,     "publishedSoap":false,     "publishedWsdl":false,
    "shared":false,     "publishedSql":false,     "executedOnServer":false,     "published4DMobile":{         "scope":"table",
"table":"Table_1"     } }
```

METHODE Lire chemin

METHODE Lire chemin (typeMéthode {; laTable}{; nomObjet{; nomObjetForm}}{; *}) -> Résultat

Paramètre	Type		Description
typeMéthode	Entier long	⇒	Sélecteur de type d'objet
laTable	Table	⇒	Référence de table
nomObjet	Texte	⇒	Nom de formulaire ou de méthode base
nomObjetForm	Texte	⇐	Nom d'objet du formulaire
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)
Résultat	Texte	⇐	Chemin complet de l'objet

Description

La commande **METHODE Lire chemin** retourne le chemin d'accès interne complet d'une méthode.

Passez dans *typeMéthode* le type de méthode dont vous souhaitez obtenir le chemin. Vous pouvez utiliser les constantes suivantes, placées dans le thème **Accès objets développement** :

Constante	Type	Valeur	Comment
Chemin formulaire projet	Entier long	4	Chemin des méthodes formulaire projet et de toutes leurs méthodes objet. Exemples : *[projectForm]/monForm/{formMethod}* *[projectForm]/monForm/bouton1* *[projectForm]/monForm/ma%2liste* *[projectForm]/monForm2/bouton1*
Chemin formulaire table	Entier long	16	Chemin des méthodes formulaire table et de toutes leurs méthodes objet. Exemples : *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/bouton1* *[tableForm]/table_1/Form1/ma%2liste* *[tableForm]/table_2/Form1/ma%2liste*
Chemin méthode base	Entier long	2	Chemin des méthodes base définies (nom anglais). Liste de ces méthodes : *[databaseMethod]/onStartup* *[databaseMethod]/onExit* *[databaseMethod]/onDrop* *[databaseMethod]/onBackupStartup* *[databaseMethod]/onBackupShutdown* *[databaseMethod]/onWebConnection* *[databaseMethod]/onWebAuthentication* *[databaseMethod]/onWebSessionSuspend* *[databaseMethod]/onServerStartup* *[databaseMethod]/onServerShutdown* *[databaseMethod]/onServerOpenConnexion* *[databaseMethod]/onServerCloseConnection* *[databaseMethod]/onSystemEvent* *[databaseMethod]/onSqlAuthentication*
Chemin Méthode projet	Entier long	1	Nom de la méthode. Exemple : *MaMethodeProjet*
Chemin trigger	Entier long	8	Chemin des triggers de la base. Exemples : *[trigger]/table_1* *[trigger]/table_2*

Passez des valeurs dans les paramètres *laTable*, *nomObjet* et *nomObjetForm* en fonction du type d'objet dont vous souhaitez récupérer le chemin d'accès de la méthode :

Type d'objet	*table*	*nomObjet*	*nomObjetForm*
Chemin Formulaire projet		X	X (optionnel)
Chemin Formulaire table	X	X	X (optionnel)
Chemin Méthode base		X	
Chemin Méthode projet		X	
Chemin Trigger	X		

Si l'objet n'est pas trouvé (type de méthode inconnu ou non valide, table manquante, etc.), une erreur est générée.

Si la commande est exécutée depuis un composant, elle retourne par défaut les chemins des méthodes du composant. Si vous passez le paramètre *, le tableau contiendra les chemins des méthodes de la base hôte.

Exemple

```
 //Récupérer le chemin d'accès de la méthode base "Sur ouverture"
$chemin:=METHODE Lire chemin(Chemin méthode base;"onStartup")

 //Récupérer le chemin d'accès du trigger de la table [Emp] :
$chemin:=METHODE Lire chemin(Chemin trigger;[Emp])

 //Récupérer le chemin d'accès de la méthode de l'objet "OK" du formulaire "input" de la table [Emp] :
$chemin:=METHODE Lire chemin(Chemin formulaire table;[Emp];"input";"OK")
```

METHODE LIRE CHEMINS

METHODE LIRE CHEMINS ({nomDossier ;} typeMéthode ; tabChemins {; marqueur}{; *})

Paramètre	Type		Description
nomDossier	Texte	➡	Nom de dossier de la page Démarrage
typeMéthode	Entier long	➡	Sélecteur de type de méthode à récupérer
tabChemins	Tableau texte	⬅	Tableau des chemins et noms des méthodes
marqueur	Variable entier long	➡	Valeur minimum de marqueur
		⬅	Nouvelle valeur courante
*	Opérateur	➡	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE LIRE CHEMINS** remplit le tableau *tabChemins* avec les chemins d'accès internes et les noms des méthodes de l'application du type défini par le paramètre *typeMéthode*.

Si votre code est organisé en "dossiers" dans l'Explorateur de 4D (page Démarrage), vous pouvez passer dans le paramètre optionnel *nomDossier* un nom de dossier. Dans ce cas, le tableau *tabChemins* ne contient que les chemins des méthodes situées à cet emplacement.

Note : Il n'est pas possible d'utiliser le caractère "@" dans *nomDossier*.

Passez dans *typeMéthode* le type de méthode dont vous souhaitez obtenir les chemins dans le tableau *tabChemins*. Vous pouvez utiliser les constantes suivantes, placées dans le thème **Accès objets développement** (vous pouvez utiliser une constante ou une combinaison de constantes) :

Constante	Type	Valeur	Comment
Chemin formulaire projet	Entier long	4	Chemin des méthodes formulaire projet et de toutes leurs méthodes objet. Exemples : *[projectForm]/monForm/{formMethod}* *[projectForm]/monForm/bouton1* *[projectForm]/monForm/ma%2liste* *[projectForm]/monForm2/bouton1*
Chemin formulaire table	Entier long	16	Chemin des méthodes formulaire table et de toutes leurs méthodes objet. Exemples : *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/bouton1* *[tableForm]/table_1/Form1/ma%2liste* *[tableForm]/table_2/Form1/ma%2liste*
Chemin méthode base	Entier long	2	Chemin des méthodes base définies (nom anglais). Liste de ces méthodes : *[databaseMethod]/onStartup* *[databaseMethod]/onExit* *[databaseMethod]/onDrop* *[databaseMethod]/onBackupStartup* *[databaseMethod]/onBackupShutdown* *[databaseMethod]/onWebConnection* *[databaseMethod]/onWebAuthentication* *[databaseMethod]/onWebSessionSuspend* *[databaseMethod]/onServerStartup* *[databaseMethod]/onServerShutdown* *[databaseMethod]/onServerOpenConnexion* *[databaseMethod]/onServerCloseConnection* *[databaseMethod]/onSystemEvent* *[databaseMethod]/onSqlAuthentication*
Chemin Méthode projet	Entier long	1	Nom de la méthode. Exemple : *MaMethodeProjet*
Chemin tous les objets	Entier long	31	Combinaison des chemins de toutes les méthodes de la base
Chemin trigger	Entier long	8	Chemin des triggers de la base. Exemples : *[trigger]/table_1* *[trigger]/table_2*

Le paramètre *marqueur* vous permet de ne récupérer que les chemins des méthodes modifiées à compter d'un instant spécifique. Dans le cadre d'un système de contrôle de version, ce principe permet de mettre à jour uniquement les méthodes modifiées depuis la dernière sauvegarde.
Le fonctionnement est le suivant : 4D maintient un compteur de modification des méthodes. A chaque fois qu'une méthode est créée ou réenregistrée, ce compteur est incrémenté et sa valeur courante est stockée dans le marqueur interne de la méthode. Si vous passez *marqueur*, la commande ne retourne que les méthodes dont le marqueur est supérieur ou égal à la valeur passée dans ce paramètre. De plus, la commande retourne dans *marqueur* la nouvelle valeur courante du compteur de modification, c'est-à-dire la valeur la plus élevée. Si vous stockez cette valeur, il vous suffira de la passer lors de l'appel suivant à la commande afin de ne récupérer que les méthodes nouvelles ou modifiées.

Si la commande est exécutée depuis un composant, elle retourne par défaut les chemins des méthodes du composant. Si vous passez le paramètre *, le tableau contiendra les chemins des méthodes de la base hôte.

Si la commande détecte un nom de méthode dupliqué, l'erreur -9802 est générée ("Chemin d'objet non unique"). Il est recommandé dans ce cas d'utiliser le CSM afin de vérifier la structure de la base de données.

Exemple 1

Récupération des méthodes projet placée dans un dossier "web" :

```
METHODE LIRE CHEMINS("web";Chemin Méthode projet;tabChemins)
```

Exemple 2

Récupération des méthodes base et des triggers :

```
METHODE LIRE CHEMINS(Chemin trigger+Chemin méthode base;tabChemins)
```

Exemple 3

Récupération des méthodes projet modifiées depuis le dernier backup :

```
 // On charge la dernière valeur stockée
$stamp:=Max([Backups]cur_stamp)
METHODE LIRE CHEMINS(Chemin Méthode projet;tabChemins;$stamp)
 // On stocke la nouvelle valeur
CREER ENREGISTREMENT([Backups])
[Backups]cur_stamp:=$stamp
STOCKER ENREGISTREMENT([Backups])
```

Exemple 4

Reportez-vous à l'exemple de la commande **METHODE FIXER CODE**.

METHODE LIRE CHEMINS FORM

METHODE LIRE CHEMINS FORM ({laTable ;} tabChemins {; filtre}{; marqueur}{; *})

Paramètre	Type		Description
laTable	Table	⇒	Référence de table
tabChemins	Tableau texte	⇐	Tableau des chemins et noms des méthodes
filtre	Texte	⇒	Filtrage des noms
marqueur	Variable entier long	⇒	Valeur minimum de marqueur
		⇐	Nouvelle valeur courante
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE LIRE CHEMINS FORM** remplit le tableau *tabChemins* avec les chemins d'accès internes et les noms des méthodes de tous les objets des formulaires, ainsi que des méthodes formulaire. Les méthodes formulaire sont libellées {formMethod}.

Seuls les objets contenant du code sont listés. Par exemple, les boutons uniquement associés à une action automatique ne sont pas retournés.

Si vous passez le paramètre *laTable*, la commande retourne les objets des formulaires table associés à cette table. Si vous omettez ce paramètre, le commande retourne les objets des formulaires projet de la base.

Vous pouvez restreindre la liste des formulaires en passant une chaîne de comparaison dans le paramètre *filtre* : dans ce cas, seuls les formulaires dont le nom correspond au filtre seront retournés. Vous pouvez utiliser le caractère @ afin de définir des filtres de type "commence par", "se termine par" ou "contient". Si vous passez une chaîne vide, le paramètre *filtre* est ignoré.

Le paramètre *marqueur* vous permet de ne récupérer que les chemins des méthodes modifiées à compter d'un instant spécifique. Dans le cadre d'un système de contrôle de version, ce principe permet de mettre à jour uniquement les méthodes modifiées depuis la denière sauvegarde.
Le fonctionnement est le suivant : 4D maintient un compteur de modification des méthodes. A chaque fois qu'une méthode est créée ou réenregistrée, ce compteur est incrémenté et sa valeur courante est stockée dans le marqueur interne de la méthode. Si vous passez *marqueur*, la commande ne retourne que les méthodes dont le marqueur est supérieur ou égal à la valeur passée dans ce paramètre. De plus, la commande retourne dans *marqueur* la nouvelle valeur courante du compteur de modification, c'est-à-dire la valeur la plus élevée. Si vous stockez cette valeur, il vous suffira de la passer lors de l'appel suivant à la commande afin de ne récupérer que les méthodes nouvelles ou modifiées.

Si la commande est exécutée depuis un composant, elle retourne par défaut les chemins des méthodes du composant. Si vous passez le paramètre *, le tableau contiendra les chemins des méthodes de la base hôte.

Note : La commande ne liste pas les objets des formulaires hérités ni des sous-formulaires.

Si la commande détecte un nom d'objet dupliqué, l'erreur -9802 est générée ("Chemin d'objet non unique"). Il est recommandé dans ce cas d'utiliser le CSM afin de vérifier la structure de la base de données.

Exemple 1

Liste de tous les objets du formulaire "input" de la table [Emp]. A noter que les méthodes formulaire table (et les méthodes formulaire projet) sont traitées comme des objets appartenant au formulaire :

```
METHODE LIRE CHEMINS FORM([Emp];tabChemins;"input")
 // Contenu de tabChemins (par exemple)
 // [tableForm]/input/{formMethod} -> Méthode formulaire
 // [tableForm]/input/bOK -> Méthode objet
 // [tableForm]/input/bCancel -> Méthode objet
```

Exemple 2

Liste des objets du formulaire projet "dial" :

```
METHODE LIRE CHEMINS FORM(tabChemins;"dial")
```

Exemple 3

Liste de tous les objets des formulaires "input" de la table [Emp] à partir d'un composant :

```
METHODE LIRE CHEMINS FORM(([Emp];tabChemins;"input@";*)
```

METHODE LIRE CODE

METHODE LIRE CODE (chemin ; code {; option} {; *})

Paramètre	Type		Description
chemin	Texte, Tableau texte	⇒	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
code	Texte, Tableau texte	⇐	Code de(s) méthode(s) désignée(s)
option	Entier long	⇒	0 ou omis = export simple (sans tokens), 1 = export avec tokens
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE LIRE CODE** retourne dans le paramètre *code* le contenu de la ou des méthode(s) désignée(s) par le paramètre *chemin*. La commande peut retourner le code de tous les types de méthodes : méthodes base, triggers, méthodes projet, méthodes formulaire et méthodes objet.

Vous pouvez utiliser deux types de syntaxes, basées soit sur des tableaux texte, soit sur des variables texte :

```
C_TEXTE(vTchemin) // variables texte
C_TEXTE(vTcode)
METHODE LIRE CODE(vTchemin;vTcode) // code d'une seule méthode
```

```
TABLEAU TEXTE(tabChemins;0) // tableaux texte
TABLEAU TEXTE(tabCodes;0)
METHODE LIRE CODE(tabChemins;tabCodes) // codes de plusieurs méthodes
```

Il n'est pas possible de mixer les deux syntaxes.

Si un chemin d'accès passé est invalide, le paramètre *code* est laissé vide et une erreur est générée.

Dans le texte du *code* généré par la commande :

- Les noms des commandes sont écrits en anglais, hormis si vous utilisez une version française de 4D et avez coché la préférence "Utiliser langage français et paramètres régionaux système" (cf. **Page Méthodes**). Le code peut contenir les *tokens* du langage afin de le rendre indépendant de la langue et de la version, si vous utilisez le paramètre *option* (cf. ci-dessous).
- Le texte est indenté avec des caractères de tabulation en fonction des structures de programmation, à l'instar de l'éditeur de méthodes, afin d'augmenter la lisibilité du code.
- Une ligne est ajoutée en en-tête du code généré, contenant des métadonnées utilisées lors de l'import du code, par exemple :

  ```
  // %attributes = {"lang":"fr","invisible":true,"folder":"Web3"}
  ```

  En cas d'import, cette ligne n'est pas importée, elle est utilisée pour définir les attributs à appliquer (les attributs non spécifiés sont remis à leur valeur par défaut). L'attribut "lang" définit la langue d'export, il permet d'empêcher un import dans une application en langue différente (dans ce cas, une erreur est générée). L'attribut "folder" contient le nom du dossier parent de la méthode, il n'apparaît pas si la méthode n'a pas de dossier parent. Des attributs supplémentaires peuvent être définis. Pour plus d'informations, reportez-vous à la description de la commande **METHODE FIXER ATTRIBUTS**.

Le paramètre *option* vous permet de sélectionner le mode d'exportation du code concernant les éléments "tokenisés" de la ou des méthode(s) :

- Si vous passez 0 ou omettez le paramètre *option*, le code de la méthode est exporté sans tokens, c'est-à-dire exactement comme affiché dans l'éditeur de méthodes.
- Si vous passez 1 ou la constante Code avec tokens, le code de la méthode est exporté avec des tokens, c'est-à-dire que les éléments "tokenisés" sont suivis de leur référence interne dans le contenu du *code* exporté. Par exemple, l'expression "**Chaine**(a)" est exportée "**Chaine**:C10(a)", où "C10" est la référence interne de la commande **Chaine**.

Les éléments tokenisés du langage incluent :

- les commandes et constantes 4D,
- les noms de tables et de champs,
- les commandes des plug-ins 4D.

Le code exporté avec ses tokens est indépendant de tout renommage ultérieur des éléments du langage. Grâce aux tokens, le code fourni sous forme de texte sera toujours correctement interprété par 4D, que ce soit via la commande **METHODE FIXER CODE** ou même le copier-coller. Pour plus d'informations sur la syntaxe tokens 4D, veuillez vous reporter à la section **Utiliser des tokens dans les formules**.

Si la commande est exécutée depuis un composant, elle s'applique par défaut aux méthodes du composant. Si vous passez le paramètre *, elle accède aux méthodes de la base hôte.

Exemple 1

Reportez-vous à l'exemple de la commande **METHODE FIXER CODE**.

Exemple 2

Cet exemple illustre les effets du paramètre *option*.

Vous voulez importer le code de la méthode "simple_init" suivante :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TOUT SELECTIONNER([Customer])
Fin de cas
```

Si vous exécutez le code suivant :

```
C_TEXTE($code)
C_TEXTE($contents)
$code:=METHODE Lire chemin(Chemin Méthode projet;"simple_init")
METHODE LIRE CODE($code;$contents;0) //pas de tokens
TEXTE VERS DOCUMENT("simple_init.txt";$contents)
```

Le document résultant contient :

```
  //%attributes = {"lang":"fr"} commentaire réservé, ajouté par 4D
Au cas ou
    : (Evenement formulaire=Sur chargement)
       TOUT SELECTIONNER([Customer])
Fin de cas
```

Si vous exécutez le code suivant :

```
C_TEXTE($code)
C_TEXTE($contents)
$code:=METHODE Lire chemin(Chemin Méthode projet;"simple_init")
METHODE LIRE CODE($code;$contents;Code avec tokens) //ajouter tokens
TEXTE VERS DOCUMENT("simple_init.txt";$contents)
```

Le document résultant contient alors :

```
  //%attributes = {"lang":"fr"} commentaire réservé, ajouté par 4D
Au cas ou
    : (Evenement formulaire:C388=Sur chargement:K2:1)
       TOUT SELECTIONNER:C47([Customer:1])
Fin de cas
```

METHODE LIRE COMMENTAIRES

METHODE LIRE COMMENTAIRES (chemin ; commentaires {; *})

Paramètre	Type		Description
chemin	Texte, Tableau texte	➡	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
commentaires	Texte, Tableau texte	⬅	Commentaires de la ou des méthode(s) désignée(s)
*	Opérateur	➡	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE LIRE COMMENTAIRES** retourne dans le paramètre *commentaires* les commentaires de la ou des méthode(s) désignée(s) par le paramètre *chemin*.

Les commentaires lus par cette commande sont ceux définis dans l'Explorateur de 4D (à ne pas confondre avec les lignes de commentaires dans le code, qui peuvent être lues à l'aide de **METHODE LIRE CODE**) :

Ces commentaires peuvent être générés uniquement pour les méthodes de type triggers, méthodes projet et méthodes formulaire. Ils contiennent du texte stylé.

Note : Les formulaires et les méthodes formulaire partagent les mêmes commentaires.

Vous pouvez utiliser deux types de syntaxes, basées soit sur des tableaux texte, soit sur des variables texte :

```
C_TEXTE(vTchemin) // variables texte
C_TEXTE(vTcommentaires)
METHODE LIRE COMMENTAIRES(vTchemin;vTcommentaires) // commentaires d'une seule méthode
```

```
TABLEAU TEXTE(tabChemins;0) // tableaux texte
TABLEAU TEXTE(tabCommentaires;0)
METHODE LIRE COMMENTAIRES(tabChemins;tabCommentaires) // commentaires de plusieurs méthodes
```

Il n'est pas possible de mixer les deux syntaxes.

Si la commande est exécutée depuis un composant, elle s'applique par défaut aux méthodes du composant. Si vous passez le paramètre *, elle accède aux méthodes de la base hôte.

METHODE LIRE DATE MODIFICATION

METHODE LIRE DATE MODIFICATION (chemin ; dateMod ; heureMod {; *})

Paramètre	Type		Description
chemin	Texte, Tableau texte	⇒	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
dateMod	Date, Tableau date	⇐	Date(s) de modification de méthode(s)
heureMod	Heure, Tableau entier long	⇐	Heure(s) de modification de méthode(s)
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE LIRE DATE MODIFICATION** retourne dans les paramètres *dateMod* et *heureMod* les dates et heures de dernière modification de la ou des méthode(s) désignée(s) par le paramètre *chemin*.

Vous pouvez utiliser deux types de syntaxes, basées soit sur des tableaux, soit sur des variables :

```
C_TEXTE(vTchemin) // variables
C_DATE(vDate)
C_HEURE(vHeure)
METHODE LIRE DATE MODIFICATION(vTchemin;vDate;vHeure) // date et heure d'une seule méthode
```

```
TABLEAU TEXTE(tabChemins;0) // tableaux
TABLEAU DATE(tabDates;0)
TABLEAU ENTIER LONG(tabHeures;0)
METHODE LIRE DATE MODIFICATION(tabChemins;tabDates;tabHeures) // dates et heures de plusieurs méthodes
```

Il n'est pas possible de mixer les deux syntaxes.

Si la commande est exécutée depuis un composant, elle s'applique par défaut aux méthodes du composant. Si vous passez le paramètre *, elle accède aux méthodes de la base hôte.

Exemple 1

Vous souhaitez connaître les dates et heures de modification de plusieurs méthodes :

```
TABLEAU TEXTE(tabChemins;0)
AJOUTER A TABLEAU(tabChemins;"MaMethode1")
AJOUTER A TABLEAU(tabChemins;"MaMethode2")
...
TABLEAU DATE(tabDates;0)
TABLEAU ENTIER LONG(tabHeures;0)
METHODE LIRE DATE MODIFICATION(tabChemins;tabDates;tabHeures)
```

Exemple 2

Vous souhaitez obtenir les dates de modification des méthodes d'un module, préfixées "Web_". Il n'est pas possible d'utiliser "@" dans le chemin, vous pouvez cependant écrire :

```
TABLEAU TEXTE($_webMethod;0)      
METHODE LIRE NOMS($_webMethod;"Web_@")
TABLEAU DATE($_date;0)
TABLEAU ENTIER LONG($_time;0)
METHODE LIRE DATE MODIFICATION($_webMethod;$_date;$_time)
```

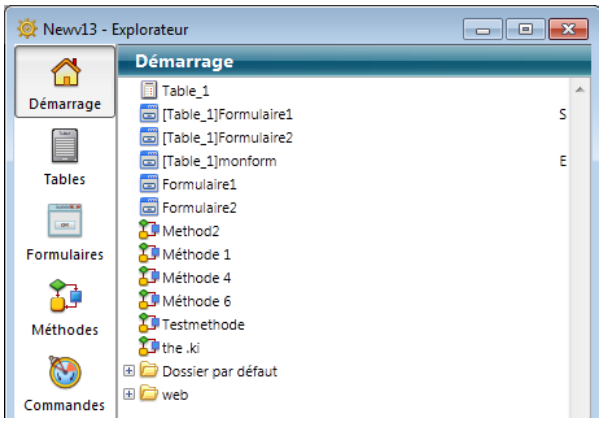
METHODE LIRE DOSSIERS

METHODE LIRE DOSSIERS (tabNoms {; filtre}{; *})

Paramètre	Type		Description
tabNoms	Tableau texte	⇐	Tableau des noms de dossiers de la page Démarrage
filtre	Texte	⇒	Filtrage des noms
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE LIRE DOSSIERS** retourne dans le tableau *tabNoms* les noms des dossiers créés dans la page Démarrage de l'Explorateur de 4D :



Comme les noms des dossiers doivent être uniques, la hiérarchie n'est pas retournée dans le tableau.

Vous pouvez restreindre la liste des dossiers en passant une chaîne de comparaison dans le paramètre *filtre* : dans ce cas, seuls les dossiers dont le nom correspond au filtre seront retournés. Vous pouvez utiliser le caractère @ afin de définir des filtres de type "commence par", "se termine par" ou "contient". Si vous passez une chaîne vide, le paramètre *filtre* est ignoré.

Si la commande est exécutée depuis un composant, elle retourne par défaut les chemins des méthodes du composant. Si vous passez le paramètre *, le tableau contiendra les chemins des méthodes de la base hôte.

METHODE LIRE NOMS

METHODE LIRE NOMS (tabNoms {; filtre}{; *})

Paramètre	Type		Description
tabNoms	Tableau texte	⇐	Tableau des noms de méthodes projet
filtre	Texte	⇒	Filtrage des noms
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE LIRE NOMS** remplit le tableau *tabNoms* avec les noms des méthodes projet créées dans l'application.

Par défaut, toutes les méthodes sont listées. Vous pouvez restreindre cette liste en passant une chaîne de comparaison dans le paramètre *filtre* : dans ce cas, seules les méthodes dont le nom correspond au filtre seront retournées. Vous devez utiliser le caractère @ afin de définir des filtres de type "commence par", "se termine par" ou "contient". Si vous passez une chaîne vide, le paramètre *filtre* est ignoré.

Si la commande est exécutée depuis un composant, elle retourne par défaut les noms des méthodes projet du composant. Si vous passez le paramètre *, le tableau contiendra les méthodes projet de la base hôte.

Note : Les méthodes placées dans la corbeille ne sont pas listées.

Exemple

Exemples d'utilisations types :

```
 // Liste de toutes les méthodes projet de la base
METHODE LIRE NOMS(t_Noms)

 // Liste des méthodes projet débutant par une chaîne spécifique
METHODE LIRE NOMS(t_Noms;"web_@")

 // Liste des méthodes projet de la base hôte débutant par une chaîne spécifique
METHODE LIRE NOMS(t_Noms;"web_@";*)
```

METHODE OUVRIR CHEMIN

METHODE OUVRIR CHEMIN (chemin {; *})

Paramètre	Type		Description
chemin	Texte	⇨	Chemin de la méthode à ouvrir
*	Opérateur	⇨	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE OUVRIR CHEMIN** ouvre, dans l'éditeur de méthodes de 4D, la méthode dont vous avez passé le chemin d'accès interne dans le paramètre *chemin*.

Cette commande peut ouvrir tous les types de méthodes (objet, formulaire, trigger, projet ou base). La méthode doit déjà exister. Si le paramètre *chemin* ne correspond pas à une méthode existante, l'erreur -9801 "Impossible d'ouvrir la méthode" est retournée.

Vous pouvez exécuter cette commande depuis un composant, mais dans ce cas vous devez passer le paramètre * car l'accès en écriture au code du composant n'est pas possible. Si vous omettez le paramètre * dans ce contexte, l'erreur -9763 est générée.

METHODE RESOUDRE CHEMIN

METHODE RESOUDRE CHEMIN (chemin ; typeMéthode ; ptrTable ; nomObjet ; nomObjetForm {; *})

Paramètre	Type		Description
chemin	Texte	⇒	Chemin à résoudre
typeMéthode	Entier long	⇐	Sélecteur de type d'objet
ptrTable	Pointeur	⇐	Référence de table
nomObjet	Texte	⇐	Nom de formulaire ou de méthode base
nomObjetForm	Texte	⇐	Nom d'objet du formulaire
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **METHODE RESOUDRE CHEMIN** analyse le chemin d'accès interne passé dans le paramètre *chemin* et retourne ses différentes composantes dans les paramètres *typeMéthode*, *ptrTable*, *nomObjet* et *nomObjetForm*.

Le paramètre *typeMéthode* retourne une valeur indiquant le type de la méthode. Vous pouvez comparer cette valeur aux constantes suivantes du thème **Accès objets développement** :

Constante	Type	Valeur	Comment
Chemin formulaire projet	Entier long	4	Chemin des méthodes formulaire projet et de toutes leurs méthodes objet. Exemples : *[projectForm]/monForm/{formMethod}* *[projectForm]/monForm/bouton1* *[projectForm]/monForm/ma%2liste* *[projectForm]/monForm2/bouton1*
Chemin formulaire table	Entier long	16	Chemin des méthodes formulaire table et de toutes leurs méthodes objet. Exemples : *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/bouton1* *[tableForm]/table_1/Form1/ma%2liste* *[tableForm]/table_2/Form1/ma%2liste*
Chemin méthode base	Entier long	2	Chemin des méthodes base définies (nom anglais). Liste de ces méthodes : *[databaseMethod]/onStartup* *[databaseMethod]/onExit* *[databaseMethod]/onDrop* *[databaseMethod]/onBackupStartup* *[databaseMethod]/onBackupShutdown* *[databaseMethod]/onWebConnection* *[databaseMethod]/onWebAuthentication* *[databaseMethod]/onWebSessionSuspend* *[databaseMethod]/onServerStartup* *[databaseMethod]/onServerShutdown* *[databaseMethod]/onServerOpenConnexion* *[databaseMethod]/onServerCloseConnection* *[databaseMethod]/onSystemEvent* *[databaseMethod]/onSqlAuthentication*
Chemin Méthode projet	Entier long	1	Nom de la méthode. Exemple : *MaMethodeProjet*
Chemin trigger	Entier long	8	Chemin des triggers de la base. Exemples : *[trigger]/table_1* *[trigger]/table_2*

Le paramètre *ptrTable* contient un pointeur sur une table de la base si le chemin référence une méthode formulaire table ou un trigger.

Le paramètre *nomObjet* contient soit :

- un nom de formulaire si le chemin référence un formulaire table ou projet
- un nom de méthode base si le chemin référence une méthode base.

Le paramètre *nomObjetForm* contient un nom d'objet de formulaire si le chemin référence une méthode objet.

Si la commande est exécutée depuis un composant, elle considère par défaut que *chemin* désigne une méthode du composant. Si vous passez le paramètre *, elle considère que *chemin* désigne une méthode de la base hôte.

Exemple 1

Résolution d'un chemin de méthode base :

```
C_ENTIER LONG($methodType)
C_POINTEUR($tablePtr)
C_TEXTE($objectName)
C_TEXTE($objectFormName)

METHODE RESOUDRE CHEMIN("[databaseMethod]/onStartup";$methodType;$tablePtr;$objectName;$objectFormName)
 // $methodType: 2
 // $tablePtr: pointeur Nil
 // $objectName: "onStartup"
 // $objectFormName: ""
```

Exemple 2

Résolution d'un chemin d'objet de méthode formulaire table :

```
C_ENTIER LONG($methodType)
C_POINTEUR($tablePtr)
C_TEXTE($objectName)
C_TEXTE($objectFormName)

METHODE RESOUDRE CHEMIN("
[tableForm]/Table1/output%2A1/myVar%2A1";$methodType;$tablePtr;$objectName;$objectFormName)
  // $methodType: 16
  // $tablePtr: -> [Table1]
  // $objectName: "output*1"
  // $objectFormName: "Btn*1"
```

BLOB

- Commandes du thème BLOB
- BLOB VERS DOCUMENT
- BLOB vers entier
- BLOB vers entier long
- BLOB vers liste
- BLOB vers reel
- BLOB vers texte
- BLOB VERS VARIABLE
- COMPRESSER BLOB
- COPIER BLOB
- CRYPTER BLOB
- DECOMPRESSER BLOB
- DECRYPTER BLOB
- DOCUMENT VERS BLOB
- ENTIER LONG VERS BLOB
- ENTIER VERS BLOB
- FIXER TAILLE BLOB
- INSERER DANS BLOB
- LIRE PROPRIETES BLOB
- LISTE VERS BLOB
- REEL VERS BLOB
- SUPPRIMER DANS BLOB
- Taille BLOB
- TEXTE VERS BLOB
- VARIABLE VERS BLOB

Définition

4D prend en charge les données de type BLOB (Binary Large OBjects).

Vous pouvez définir des champs de type BLOB, des variables de type BLOB et des tableaux de type BLOB :

- Pour créer un champ de type BLOB, sélectionnez BLOB dans la liste déroulante **Type de champ** dans la fenêtre des **Propriétés de champ**.
- Pour créer une variable de type BLOB, utilisez la directive de compilation **C_BLOB**. Vous pouvez créer des variables BLOB locales, process et interprocess.
- Pour créer un tableau de type BLOB, utilisez la commande **TABLEAU BLOB**.

Dans 4D, un BLOB est une série contiguë d'octets de longueur variable qui peut être traitée comme un seul objet ou dont les octets peuvent être adressés individuellement. Un BLOB peut être vide (longueur nulle) ou contenir jusqu'à 2147483647 octets (2 Go).

Les BLOBs et la mémoire

Lorsque vous travaillez avec un BLOB, il est stocké entièrement en mémoire. Si vous travaillez avec une variable ou un tableau, le BLOB n'existe qu'en mémoire. Si vous travaillez avec un champ de type BLOB, il est chargé en mémoire à partir du disque, comme le reste de l'enregistrement auquel il appartient.

A l'instar des autres types de champs pouvant contenir une grande quantité de données (comme les champs de type Image), les champs de type BLOB ne sont pas dupliqués en mémoire lorsque vous modifiez un enregistrement. Par conséquent, les résultats renvoyés par **Ancien** et **Modifie** ne sont pas significatifs lorsque ces fonctions sont appliquées à des champs de type BLOB.

Afficher des BLOBs

Comme un BLOB peut contenir n'importe quel type de données, il n'existe pas de mode de représentation à l'écran par défaut des BLOBs. Si vous affichez un champ ou une variable de type BLOB dans un formulaire, l'objet sera toujours vide, quel que soit son contenu.

Champs de type BLOB

Vous pouvez utiliser des champs de type BLOB pour stocker tout type de données dont la taille est inférieure ou égale à 2 Giga-octets. Vous ne pouvez pas indexer un champ de type BLOB. Si vous voulez rechercher des enregistrements à partir d'une valeur contenue dans un champ BLOB, il sera nécessaire d'écrire une formule.

Passage des paramètres, pointeurs et résultats de fonctions

Les BLOBs dans 4D peuvent être passés comme paramètres aux commandes 4D ou aux routines des plug-ins qui attendent un paramètre de type BLOB. Les BLOBs peuvent également être passés aux méthodes que vous créez ou être retournés comme un résultats de fonctions. Pour passer un BLOB à une de vos méthodes, vous pouvez aussi définir un pointeur vers le BLOB et passer le pointeur comme paramètre. Voici quelques exemples :

```
  ` Déclarer une variable de type BLOB
C_BLOB(touteVarBLOB)
  ` Le BLOB est passé comme paramètre à une commande 4D
FIXER TAILLE BLOB(touteVarBLOB;1024*1024)
  ` Le BLOB est passé comme paramètre à une routine externe
$CodeErr:=Faites_Quelque_chose_avec_ce_BLOB(touteVarBLOB)
  ` Le BLOB est passé comme paramètre à une méthode qui retourne un BLOB
C_BLOB(recupBlob)
recupBlob:=Remplir_Blob(touteVarBLOB)
  ` Un pointeur vers le BLOB est passé comme paramètre à une de vos méthodes
REMPLIR BLOB AVEC DES ZEROS(->touteVarBLOB)
```

Note pour les développeurs de plug ins 4D : Un paramètre de type BLOB se déclare “&O” (la lettre “O” et non le chiffre “0”).

Affectation

Vous pouvez affecter la valeur d'un BLOB à d'autres BLOBs, comme dans l'exemple suivant :

```
  ` Déclarer deux variables de type BLOB
C_BLOB(vBlobA;vBlobB)
  ` Fixer la taille du premier BLOB à 10Ko
FIXER TAILLE BLOB(vBlobA;10*1024)
  ` Affectez le premier BLOB au second
vBlobB:=vBlobA
```

En revanche, il n'existe pas d'opérateur pouvant être utilisé avec des BLOBs ; il n'existe pas d'expression de type BLOB.

Adresser le contenu d'un BLOB

Chaque octet d'un BLOB peut être adressé individuellement, à l'aide des accolades *{...}*. Dans un BLOB, les octets sont numérotés de *0* à **N-1**, **N** étant la taille du BLOB. Voici un exemple :

```
  ` Déclarer une variable de type BLOB
C_BLOB(vBlob)
  ` Fixer la taille du BLOB à 256 octets
FIXER TAILLE BLOB(vBlob;256)
  ` La boucle suivante initialise les 256 octets du BLOB à zéro
Boucle(vOctet;0;Taille BLOB(vBlob)-1)
   vBlob{vOctet}:=0
Fin de boucle
```

Comme vous pouvez adresser individuellement tous les octets d'un BLOB, vous pouvez littéralement stocker tout ce que vous voulez dans une variable ou un champ de type BLOB.

Routines 4D pour manipuler les BLOBs

4D fournit les routines suivantes pour travailler avec les BLOBs :

- **FIXER TAILLE BLOB** redimensionne la taille d'un champ ou d'une variable de type BLOB.
- **Taille BLOB** retourne la taille d'un champ ou d'une variable de type BLOB.
- **DOCUMENT VERS BLOB** et **BLOB VERS DOCUMENT** vous permettent de charger et d'écrire un document entier dans un champ ou une variable de type BLOB et inversement (en option, les data forks et les resource forks sur Macintosh).
- **VARIABLE VERS BLOB** et **BLOB VERS VARIABLE**, ainsi que **LISTE VERS BLOB** et **BLOB vers liste** vous permettent de stocker et de charger des variables 4D dans des BLOBs.
- **COMPRESSER BLOB**, **DECOMPRESSER BLOB** et **LIRE PROPRIETES BLOB** vous permettent de travailler avec des BLOBs compressés.
- Les commandes **BLOB vers entier**, **BLOB vers entier long**, **BLOB vers reel**, **BLOB vers texte**, **ENTIER VERS BLOB**, **ENTIER LONG VERS BLOB**, **REEL VERS BLOB** et **TEXTE VERS BLOB** vous permettent de manipuler et de structurer les données en provenance du disque, des ressources, du système d'exploitation, etc.
- **SUPPRIMER DANS BLOB**, **INSERER DANS BLOB** et **COPIER BLOB** permettent de gérer les gros volumes de données à l'intérieur des BLOBs.
- **CRYPTER BLOB** et **DECRYPTER BLOB** permettent de crypter et de décrypter des données dans vos bases 4D.

Ces commandes sont décrites dans ce chapitre. De plus, vous disposez des routines suivantes :

- **C_BLOB** déclare une variable de type BLOB (reportez-vous à la section **Compilateur**).
- **TABLEAU BLOB** crée ou redimensionne un tableau de type BLOB (reportez-vous à la section **Tableaux**).
- **LIRE DONNEES CONTENEUR** et **AJOUTER DONNEES AU CONTENEUR** vous permettent de gérer tout type de données stockées dans le presse-papiers (référez-vous à la section **Gestion du conteneur de données**).
- **LIRE RESSOURCE** et **_o_ÉCRIRE RESSOURCE** vous permettent de gérer tout type de données stockées dans une ressource qui se trouve sur disque (référez-vous à la section **Ressources**).
- **WEB ENVOYER BLOB** vous permet d'envoyer tout type de données à un navigateur Web. Cette commande est incluse dans le thème Serveur Web.
- **IMAGE VERS BLOB**, **BLOB VERS IMAGE** et **IMAGE VERS GIF** vous permettent d'ouvrir et de convertir des images via des BLOBs. Ces commandes sont incluses dans le thème **Images**.
- **GENERER CLES CRYPTAGE** et **GENERER DEMANDE CERTIFICAT** permettent d'exploiter le protocole SSL (Secured Socket Layer) pour crypter des données ou les connexions Web. Ces commandes sont incluses dans le thème **Utiliser le protocole TLS**.

BLOB VERS DOCUMENT

BLOB VERS DOCUMENT (document ; blob {; *})

Paramètre	Type		Description
document	Chaîne	⇒	Nom du document
blob	BLOB	⇒	Nouveau contenu du document
*	Opérateur	⇒	Macintosh seulement : la resource fork est écrite si * est passé ; sinon la data fork est écrite

Description

BLOB VERS DOCUMENT écrit le contenu de *document* en utilisant les données stockées dans *blob*.
Vous pouvez passer dans *document* le nom d'un document existant ou non. Si le *document* n'existe pas, la commande le crée. Si vous passez le nom d'un document existant, assurez-vous qu'il n'est pas déjà ouvert, sinon une erreur est générée. Si vous voulez que l'utilisateur choisisse le document, appelez les routines **Ouvrir document** ou **Creer document** et utilisez la variable système *Document* (cf. exemple ci-dessous).

Notes pour les utilisateurs Mac OS :

- Les documents Macintosh peuvent être composés de deux éléments, la resource fork et la data fork ("partie des ressources" et "partie des données"). Par défaut, la commande **BLOB VERS DOCUMENT** réécrit la data fork du document. Si vous voulez réécrire la resource fork du document, passez le paramètre optionnel *. Sous Windows, le paramètre * est ignoré.
- Les documents générés par cette commande n'ont pas de "type". Si vous souhaitez créer un document avec type, vous devez utiliser la commande **CHANGER TYPE DOCUMENT**.

Exemple

Notre exemple est une base qui permet de stocker et de rechercher rapidement des documents. Dans un formulaire entrée, vous créez un bouton vous permettant de sauvegarder un document de votre choix qui contient des données provenant d'un champ de type BLOB. La méthode de ce bouton peut être la suivante :

```
$vhRefDoc:=Ouvrir document("") ` Sélectionner un document
Si(OK=1) ` Si un document a été choisi
   FERMER DOCUMENT($vhRefDoc) ` Nous ne voulons pas qu'il reste ouvert
   BLOB VERS DOCUMENT(Document;[VotreTable]VotreChampBLOB) ` Ecrire le contenu du document
   Si(OK=0)
 ` Gérer l'erreur
   Fin de si
Fin de si
```

Variables et ensembles système

La variable système OK prend la valeur 1 si le document est correctement écrit. Sinon, elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

- Si vous essayez de réécrire un document qui est déjà ouvert par un autre process ou une autre application, une des *Erreurs du gestionnaire de fichiers du système* sera générée.
- L'espace sur disque peut être insuffisant pour l'écriture du contenu du document.
- Une erreur d'Entrée/Sortie peut survenir si le document est verrouillé, s'il se trouve sur un volume verrouillé ou si un problème survient lors de l'écriture du document.

Dans tous les cas, vous pouvez gérer les erreurs en utilisant la commande **APPELER SUR ERREUR**.

BLOB vers entier

BLOB vers entier (blob ; ordreOctet {; offset}) -> Résultat

Paramètre	Type		Description
blob	BLOB	➡	BLOB duquel obtenir la valeur entière
ordreOctet	Entier long	➡	0 Ordre d'octets mode natif 1 Ordre d'octets Macintosh 2 Ordre d'octets PC
offset	Variable	➡	Offset (en octets) dans le BLOB
		⬅	Nouvel offset après la lecture
Résultat	Entier	↩	Valeur entière (2 octets)

Description

BLOB vers entier retourne une valeur entière (2 octets) lue dans le BLOB *blob*.

Le paramètre *ordreOctet* fixe l'ordre des octets ("byte ordering") de la valeur entière à lire. Vous pouvez passer une des constantes fournies par 4D :

Constante	Type	Valeur
Ordre octets Macintosh	Entier long	1
Ordre octets natif	Entier long	0
Ordre octets PC	Entier long	2

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous ne passez pas de variable dans le paramètre optionnel *offset*, les deux premiers octets de BLOB sont lus.

Si vous passez une variable dans le paramètre optionnel *offset*, la valeur entière sur 2 octets est lue depuis l'offset exprimé en octets (à partir de zéro) du BLOB.

Note : Vous devez passer un offset compris entre 0 (zéro) et la taille du BLOB moins 2. Sinon, une erreur -111 est générée.

Après l'exécution de la commande, la variable *offset* est incrémentée du nombre d'octets qui a été lu. Vous pouvez donc réutiliser la même variable avec une autre commande de lecture de BLOBs pour lire une autre valeur placée juste après celle que vous venez de lire.

Exemple

L'exemple suivant lit 20 valeurs entières d'un BLOB à partir de l'offset 0x200 :

```
$vlOffset:=0x200
Boucle($viBoucle;0;19)
   $viValeur:=BLOB vers entier(vxUnBlob;Ordre octets PC;$vlOffset)
   ` Faire quelque chose avec $viValeur
Fin de boucle
```

BLOB vers entier long

BLOB vers entier long (blob ; ordreOctet {; offset}) -> Résultat

Paramètre	Type		Description
blob	BLOB	➡	BLOB duquel extraire la valeur de type Entier long
ordreOctet	Entier long	➡	0 = Ordre d'octets natif 1 = Ordre d'octets Macintosh 2 = Ordre d'octets PC
offset	Variable	➡	Offset (en octets) dans le BLOB
		⬅	Nouvel offset après lecture
Résultat	Entier long	↩	Valeur de type Entier long (4 octets)

Description

La fonction **BLOB vers entier long** retourne une valeur de type Entier long (4 octets) lue dans le BLOB *blob*.

Le paramètre *ordreOctet* fixe l'ordre des octets ("byte ordering") de la valeur Entier long à lire. Vous passez une des constantes fournies par 4D :

Constante	Type	Valeur
Ordre octets Macintosh	Entier long	1
Ordre octets natif	Entier long	0
Ordre octets PC	Entier long	2

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous ne passez pas de variable dans le paramètre optionnel *offset*, les quatre premiers octets de BLOB sont lus.

Si vous passez une variable dans le paramètre optionnel *offset*, l'entier long sur 4 octets est lu depuis l'offset exprimé en octets (à partir de zéro) du BLOB.

Note : Vous devez passer un offset compris entre 0 (zéro) et la taille du BLOB moins 4. Sinon, une erreur -111 est générée.

Après l'exécution de la commande, la variable *offset* est incrémentée du nombre d'octets qui a été lu. Vous pouvez donc réutiliser la même variable avec une autre commande de lecture de BLOBs pour lire une autre valeur placée juste après celle que vous venez de lire.

Exemple

L'exemple suivant lit 20 valeurs de type Entier long dans un BLOB, à partir de l'offset 0x200 :

```
$vlOffset:=0x200
Boucle($viBoucle;0;19)
   $vlValeur:=BLOB vers entier long(vxUnBlob;Ordre octets PC;$vlOffset)
   ` Faire quelque chose avec $vlValeur
Fin de boucle
```

BLOB vers liste

BLOB vers liste (blob {; offset}) -> Résultat

Paramètre	Type		Description
blob	BLOB	➡	BLOB contenant la liste hiérarchique
offset	Entier long	➡	Offset (en octets) dans le BLOB
		⬅	Nouvel offset après la lecture
Résultat	RefListe	↩	Référence de la liste nouvellement créée

Description

BLOB vers liste crée une nouvelle liste hiérarchique avec les données stockées dans le BLOB *blob* à l'offset d'octet (à partir de zéro) spécifié par *offset* et retourne un numéro de référence de liste hiérarchique pour cette nouvelle liste.

Les données présentes dans le BLOB doivent être compatibles avec la commande : généralement, vous utilisez des BLOBs préalablement remplis avec la commande **LISTE VERS BLOB**.

Si vous ne passez pas le paramètre optionnel *offset*, les valeurs de la liste sont lues à partir du début du BLOB. Si vous gérez un BLOB dans lequel plusieurs variables ou listes ont été stockées, vous devez passer le paramètre *offset* ainsi qu'une variable numérique. Avant l'appel, fixez cette variable numérique à l'offset désiré. Après l'appel, cette même variable numérique retourne l'offset de la variable suivante stockée dans le BLOB.

Après l'appel, la variable OK prend la valeur 1 si la liste hiérarchique a été correctement créée. Si l'opération ne peut pas être effectuée à cause, par exemple, d'un manque de mémoire, la variable OK prend la valeur 0.

Note pour l'indépendance de plate-forme : BLOB vers liste et **LISTE VERS BLOB** utilisent un format interne 4D pour gérer les listes stockées dans des BLOBs. L'avantage est que vous n'avez pas besoin de vous soucier de la conversion des octets ("byte swapping") entre les plates-formes lorsque vous utilisez ces deux commandes. Autrement dit, avec ces commandes, un BLOB créé sous Windows peut être réutilisé sous Mac OS et vice-versa.

Exemple

Dans l'exemple suivant, la méthode d'un formulaire entrée extrait une liste d'un champ BLOB avant que le formulaire ne s'affiche puis le stocke dans le champ BLOB lorsque la saisie est validée :

```
   ` Méthode du formulaire [Choses à Faire];"Entrée"

Au cas ou

   :(Evenement formulaire=Sur chargement)
      hListe:=BLOB vers liste([Choses à Faire]Idées)
      Si(OK=0)
         hListe:=Nouvelle liste
      Fin de si

   :(Evenement formulaire=Sur libération)
      SUPPRIMER LISTE(hListe;*)

   :(bValider=1)
      LISTE VERS BLOB(hListe;[Choses à Faire]Idées)

Fin de cas
```

Variables et ensembles système

La variable OK prend la valeur 1 si la liste a été correctement créée, sinon elle prend la valeur 0.

BLOB vers reel

BLOB vers reel (blob ; formatRéel {; offset}) -> Résultat

Paramètre	Type		Description
blob	BLOB	⇒	BLOB duquel extraire la valeur de type Réel
formatRéel	Entier long	⇒	0 Format réel natif 1 Format réel étendu 2 Format réel double Macintosh 3 Format réel double Windows
offset	Variable	⇒	Offset (en octets) dans le BLOB
		⇐	Nouvel offset après lecture
Résultat	Réel	⇒	Valeur de type Réel

Description

La fonction **BLOB vers reel** retourne une valeur de type Réel lue dans le BLOB *blob*.

Le paramètre *formatRéel* fixe le format interne et l'ordre des octets ("byte ordering") de la valeur de type Réel à lire. Vous passez une des constantes fournies par 4D :

Constante	Type	Valeur
Format réel double Macintosh	Entier long	2
Format réel double PC	Entier long	3
Format réel étendu	Entier long	1
Format réel natif	Entier long	0

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous ne passez pas de variable dans le paramètre optionnel *offset*, les 8 ou 10 premiers octets de BLOB sont lus.

Si vous passez une variable dans le paramètre optionnel *offset*, la valeur réelle est lue depuis l'offset exprimé en octets (à partir de zéro) du BLOB.

Note : Vous devez passer un offset compris entre 0 (zéro) et la taille du BLOB moins 8 ou 10. Sinon, une erreur -111 est générée.

Après l'exécution de la commande, la variable *offset* est incrémentée du nombre d'octets qui a été lu. Vous pouvez donc réutiliser la même variable avec une autre commande de lecture de BLOBs pour lire une autre valeur placée juste après celle que vous venez de lire.

Exemple

L'exemple suivant lit 20 valeurs réelles dans un BLOB à partir de l'offset 0x200 :

```
$vlOffset:=0x200
Boucle($viBoucle;0;19)
   $vrValeur:=BLOB vers reel(vxUnBlob;Format réel double PC;$vlOffset)
   ` Faire quelque chose avec $vrValeur
Fin de boucle
```

BLOB vers texte

BLOB vers texte (blob ; formatTexte {; offset {; longueurTexte}}) -> Résultat

Paramètre	Type		Description
blob	BLOB	⇨	BLOB duquel extraire le texte
formatTexte	Entier long	⇨	Format et jeu de caractères du texte
offset	Variable	⇨	Offset (en octets) dans le BLOB
		⇦	Nouvel offset après la lecture
longueurTexte	Entier long	⇨	Nombre de caractères à lire
Résultat	Texte	⇩	Texte extrait

Description

La fonction **BLOB vers texte** retourne une valeur de type Texte lue dans le BLOB *blob*.

Le paramètre *formatTexte* définit le format interne et le jeu de caractères de la valeur de type Texte à lire. Dans les bases de données créées à compter de la version 11, 4D utilise par défaut le jeu de caractères Unicode (UTF8) pour la gestion des textes. Par compatibilité, cette commande permet de “forcer” l'utilisation du jeu de caractères Mac Roman (jeu de caractères utilisé dans les versions précédentes de 4D). Le choix du jeu de caractères s’effectue via le paramètre *formatTexte*. Pour cela, passez dans *formatTexte* une des constantes suivantes, placées dans le thème **BLOB** :

Constante	Type	Valeur
Mac chaîne en C	Entier long	0
Mac chaîne pascal	Entier long	1
Mac texte avec longueur	Entier long	2
Mac texte sans longueur	Entier long	3
UTF8 chaîne en C	Entier long	4
UTF8 texte avec longueur	Entier long	5
UTF8 texte sans longueur	Entier long	6

Notes

- Les constantes “UTF8” sont utilisables uniquement lorsque l’application fonctionne en mode Unicode.
- Les constantes “Mac” ne permettent pas de traiter des textes de plus de 32 ko.
- Si vous souhaitez manipuler des jeux de caractères autres que UTF8, utilisez la commande **Convertir vers texte**. Pour plus d'informations sur ces constantes et les formats qu'elles représentent, reportez-vous à la description de la commande **TEXTE VERS BLOB**.

ATTENTION : Le nombre de caractères à lire est déterminé par le paramètre *formatTexte*, SAUF dans le cas des formats Mac texte sans longueur et UTF8 texte sans longueur pour lesquels vous devez spécifier le nombre de caractères à lire dans le paramètre *longueurTexte*. Pour les autres formats, *longueurTexte* est ignoré et vous pouvez l'omettre.

Si vous ne passez pas de variable dans le paramètre optionnel *offset*, les premiers octets de BLOB sont lus, en fonction de la valeur passée dans *formatTexte*. Notez que vous devez passer une variable dans le paramètre *offset* lorsque vous lisez une valeur de type Texte sans longueur.

Si vous passez une variable dans le paramètre optionnel *offset*, la valeur de type Texte est lue depuis l'offset exprimé en octets (à partir de zéro) du BLOB.

Note : Vous devez passer un offset compris entre 0 (zéro) et la taille du BLOB moins la taille du texte à extraire. Sinon, le résultat de la fonction ne sera pas exploitable.

Après l'exécution de la commande, la variable *offset* est incrémentée du nombre d'octets qui a été lu. Vous pouvez donc réutiliser la même variable avec une autre commande de lecture de BLOBs pour lire une autre valeur placée juste après celle que vous venez de lire.

BLOB VERS VARIABLE

BLOB VERS VARIABLE (blob ; variable {; offset})

Paramètre	Type		Description
blob	BLOB	⇒	BLOB contenant une ou plusieurs variable(s) 4D
variable	Variable	⇐	Variable à écrire avec le contenu de BLOB
offset	Entier long	⇒	Position de la variable dans BLOB
		⇐	Position de la variable suivante dans BLOB

Description

BLOB VERS VARIABLE réécrit la variable *variable* avec les données stockées dans le BLOB *blob* à l'offset d'octet (à partir de zéro) spécifié par *offset*.

Les données dans le BLOB doivent être compatibles avec la variable de destination : vous utiliserez généralement des BLOBs que vous avez précédemment remplis à l'aide de **VARIABLE VERS BLOB**.

Si vous ne spécifiez pas le paramètre *offset*, les données de la variable sont lues à partir du début du BLOB. Si le BLOB contient plusieurs variables, vous devez passer le paramètre *offset* ainsi qu'une variable numérique. Avant d'appeler la commande, définissez cette variable numérique avec l'offset correspondant. Après l'appel, la même variable numérique retourne l'offset de la variable suivante stockée dans le BLOB.

Note: BLOB VERS VARIABLE prend en charge les variables objet de type **C_OBJET**. Pour plus d'informations, reportez-vous à la commande **VARIABLE VERS BLOB**.

La variable OK prend la valeur 1 si l'opération s'est correctement déroulée. Si l'opération n'a pas pu être effectuée, par exemple à cause d'un manque de mémoire, la variable OK prend la valeur 0.

Note sur l'indépendance de plate-forme : BLOB VERS VARIABLE et **VARIABLE VERS BLOB** utilisent un format interne à 4D pour gérer les variables stockées dans les BLOBs. Vous n'avez donc pas besoin de vous préoccuper de la conversion des octets ("byte swapping") entre les différentes plates-formes lors de l'utilisation de ces deux commandes. Un BLOB créé sous Windows à l'aide de ces deux commandes peut être réutilisé sans la moindre manipulation sous Mac OS et vice-versa.

Exemple

Référez-vous aux exemples de **VARIABLE VERS BLOB**.

Variables et ensembles système

La variable OK prend la valeur 1 si la variable a été correctement réécrite, sinon elle prend la valeur 0.

COMPRESSER BLOB

COMPRESSER BLOB (blob {; compression})

Paramètre	Type		Description
blob	BLOB	⇒	BLOB à compresser
compression	Entier long	⇒	Si ce paramètre est passé : 1= taux de compression maximum 2 = vitesse de compression maximum

Description

COMPRESSER BLOB compresse le BLOB *blob* à l'aide d'un algorithme de compression.

Le paramètre optionnel *compression* vous permet de fixer la façon dont le BLOB sera compressé. Passez dans ce paramètre une des constantes suivantes, placées dans le thème **BLOB** :

Constante	Type	Valeur	Comment
GZIP méthode de compression compacte	Entier long	-1	Compression GZIP la plus compacte (au détriment de la vitesse à laquelle la compression et la décompression sont effectuées)
GZIP méthode de compression rapide	Entier long	-2	Compression/décompression GZIP la plus rapide (au détriment du taux de compression)
Méthode de compression compacte	Entier long	1	Compression interne la plus compacte (au détriment de la vitesse à laquelle la compression et la décompression sont effectuées). Méthode par défaut.
Méthode de compression rapide	Entier long	2	Compression/décompression interne la plus rapide au détriment du taux de compression (une fois compressé, le BLOB prend plus de place)

Si vous passez une autre valeur ou si vous omettez le paramètre *compression*, la méthode de compression 1 est utilisée (algorithme interne compact).

Note : La commande compresse uniquement les BLOBs de taille supérieure ou égale à 255 octets.

Après que cette commande ait été appelée, la variable système OK prend la valeur 1 si le BLOB a été correctement compressé.
Si la compression n'a pu être effectuée, OK prend la valeur 0. Dans ce cas, si l'erreur provient du fait que la taille du BLOB est inférieure à 255 octets ou que la mémoire disponible est insuffisante pour effectuer l'opération, aucune erreur n'est générée, la méthode poursuit son exécution.
En revanche, si l'erreur est causée par un problème plus important (le BLOB est endommagé), l'erreur -10600 est générée. Cette erreur, relativement rare, peut être interceptée à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

Lorsqu'un BLOB a été compressé, vous pouvez le décompresser à l'aide de la commande **DECOMPRESSER BLOB**.

Pour savoir si un BLOB a été compressé, utilisez la commande **LIRE PROPRIETES BLOB**.

ATTENTION : Un BLOB compressé est toujours un BLOB, rien ne vous empêche donc de modifier son contenu. Cependant, si vous le modifiez, la commande **DECOMPRESSER BLOB** ne pourra plus décompresser correctement le BLOB.

Exemple 1

L'exemple suivant teste si le BLOB *vxMonBlob* est compressé et, sinon, le compresse :

```
LIRE PROPRIETES BLOB(vxMonBlob;$vlCompressé;$vlTailleDécompressée;$vlTailleCourante)
Si($vlCompressé=Non compressé)
   COMPRESSER BLOB(vxMonBlob)
Fin de si
```

Notez que si vous appliquez **COMPRESSER BLOB** à un BLOB déjà compressé, la commande le détecte et ne fait rien.

Exemple 2

L'exemple suivant vous permet de sélectionner un document puis de le compresser :

```
$vhDocRef :=Ouvrir document("")
Si(OK=1)
   FERMER DOCUMENT($vhDocRef)
   DOCUMENT VERS BLOB(Document;vxBlob)
   Si(OK=1)
      COMPRESSER BLOB(vxBlob)
      Si(OK=1)
         BLOB VERS DOCUMENT(Document;vxBlob)
      Fin de si
   Fin de si
Fin de si
```

Exemple 3

Envoi de données HTTP brutes compressées en GZIP :

```
COMPRESSER BLOB($blob;GZIP méthode de compression compacte)
C_TEXTE($vEncoding)
$vEncoding:="Content-encoding: gzip"
WEB FIXER ENTETE HTTP($vEncoding)
WEB ENVOYER DONNEES($blob ;*)
```

Variables et ensembles système

La variable OK prend la valeur 1 si le BLOB a été correctement compressé, sinon elle prend la valeur 0.

COPIER BLOB

COPIER BLOB (srcBLOB ; dstBLOB ; srcOffset ; dstOffset ; nombre)

Paramètre	Type		Description
srcBLOB	BLOB	⇒	BLOB source
dstBLOB	BLOB	⇒	BLOB de destination
srcOffset	Entier long	⇒	Position dans la source pour la copie
dstOffset	Entier long	⇒	Position dans la destination pour la copie
nombre	Entier long	⇒	Nombre d'octets à copier

Description

COPIER BLOB copie le nombre d'octets spécifié par *nombre* du BLOB *srcBLOB* vers le BLOB *dstBLOB*.

La copie commence à la position (exprimée par rapport à l'origine du BLOB source) définie par *srcOffset* et est placée à partir de la position (exprimée par rapport à l'origine du BLOB de destination) définie par *dstOffset*.

Notez que le BLOB de destination peut être redimensionné si nécessaire.

CRYPTER BLOB

CRYPTER BLOB (aCrypter ; cléPrivEmetteur {; cléPubRécepteur})

Paramètre	Type		Description
aCrypter	BLOB	⇒	Données à crypter
		⇐	Données cryptées
cléPrivEmetteur	BLOB	⇒	Clé privée de l'émetteur
cléPubRécepteur	BLOB	⇒	Clé publique du récepteur

Description

La commande **CRYPTER BLOB** permet de crypter le contenu du BLOB *aCrypter* à l'aide de la clé privée de l'émetteur *cléPrivEmetteur* ainsi que, optionnellement, de la clé publique du récepteur *cléPubRécepteur*. Pour obtenir une paire de clés de cryptage (clé publique et clé privée), utilisez la routine **GENERER CLES CRYPTAGE**, placée dans le thème "Protocole sécurisé".

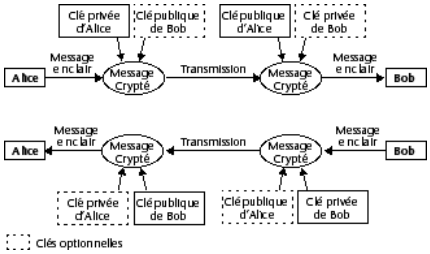
Note : La commande **CRYPTER BLOB** exploite l'algorithme et les fonctions de cryptage du protocole SSL. Par conséquent, pour pouvoir utiliser cette commande, vous devez veiller à ce que les composants nécessaires au fonctionnement du protocole SSL soient installés sur la machine — même si vous ne souhaitez pas utiliser SSL dans le cadre de connexions à un serveur Web 4D. Pour plus d'informations, reportez-vous à la section **Utiliser le protocole TLS**.

- L'utilisation d'une seule clé pour le cryptage (clé privée de l'émetteur) garantit l'impossibilité pour toute personne ne disposant pas de la clé publique de lire les données. Elle garantit également que c'est bien l'émetteur qui a crypté les données.
- L'utilisation d'une paire de clés pour le cryptage (clé privée de l'émetteur + clé publique du récepteur) garantit en outre qu'un seul récepteur pourra lire les données.

Le format interne des BLOBs contenant des clés est le PKCS. Ce format standard, multi-plate-forme, permet l'échange ou la manipulation des clés par simple copier-coller dans un Email ou un fichier texte.

Après l'exécution de la commande, le BLOB *aCrypter* contient les données cryptées. Ces données ne pourront être décryptées qu'avec la commande **DECRYPTER BLOB**, à laquelle la clé publique de l'émetteur sera passée en paramètre. En outre, si la clé publique (optionnelle) du récepteur avait été utilisée pour le cryptage, la clé privée du récepteur sera également nécessaire pour le décryptage.

Principe du cryptage à clés publiques/privées pour l'échange de messages entre deux individus, "Alice" et "Bob"



Note : L'algorithme de cryptage comporte une fonction de vérification d'intégrité (checksum), afin d'empêcher toute modification malveillante ou accidentelle du contenu du BLOB. Par conséquent, un BLOB crypté ne doit pas être modifié, sous peine de ne pas pouvoir être décrypté.

Optimisation des commandes de cryptage

Le cryptage des données ralentit l'exécution de l'application, en particulier si une paire de clés est utilisée. Deux types d'optimisations sont toutefois possibles :

- Suivant la quantité de mémoire disponible, la commande s'exécute en mode "synchrone" ou "asynchrone".
  Le mode asynchrone est plus rapide, car il ne bloque pas les autres process. Ce mode est automatiquement utilisé si la mémoire disponible est au moins égale à 2 fois la taille de la source à crypter.
  Dans le cas contraire, pour des raisons de sécurité, le mode synchrone est utilisé. Ce mode est plus lent car les autres process sont bloqués.
- Dans le cas de BLOBs volumineux, l'astuce consiste à crypter uniquement une partie déterminée et sensible du BLOB, afin de réduire la taille des données à traiter et donc le temps d'exécution.

Exemple

• Utilisation d'une seule clé

Une société veut garantir la confidentialité d'informations stockées dans une base 4D. Elle doit régulièrement envoyer ces données à ses filiales, par exemple sous la forme de fichiers via Internet.

1. La société commence par générer une paire de clés à l'aide de la commande **GENERER CLES CRYPTAGE**.

```
`Méthode GENERE_CLES_TXT
C_BLOB($BcléPublique;$BcléPrivée)
GENERER CLES CRYPTAGE($BcléPrivée;$BcléPublique)
BLOB VERS DOCUMENT("cléPublique.txt";$BcléPublique)
BLOB VERS DOCUMENT("cléPrivée.txt";$BcléPrivée)
```

2. La société conserve la clé privée, et remet à chaque filiale une copie du document contenant la clé publique. Il faut, bien entendu, que cette transmission s'effectue d'une façon sûre, par exemple par la copie sur une disquette remise physiquement aux filiales.

3. Par la suite, la société copie les informations confidentielles (stockées par exemple dans un champ texte) dans des BLOBs et les crypte avec sa clé privée :

```
`Méthode CRYPTER_INFOS
C_BLOB($vbCrypté;$vbcléPrivée)
C_TEXTE($vtCrypter)
```

```
$vtCrypter:=[Confidentiel]Info
VARIABLE VERS BLOB($vtCrypter;$vbCrypté)
DOCUMENT VERS BLOB("cléPrivée.txt";$vbcléPrivée)
Si(OK=1)
   CRYPTER BLOB($vbCrypté;$vbcléPrivée)
   BLOB VERS DOCUMENT("MiseAJour.txt";$vbCrypté)
Fin de si
```

4. Le fichier de mise à jour peut alors être envoyé aux filiales (même en passant par un canal non sécurisé comme Internet). Si un tiers intercepte le fichier crypté, il sera dans l'incapacité de le décrypter sans la clé publique.

5. Chaque filiale peut, quant à elle, décrypter le document à l'aide de la clé publique :

```
 `Méthode DECRYPTER_INFOS
C_BLOB($vbCrypté;$vbcléPublique)
C_TEXTE($vtDécrypté)
C_HEURE($vhRefDoc)

ALERTE("Veuillez sélectionner le document crypté.")
$vhRefDoc:=Ouvrir document("") `Sélection du fichier MiseAJour.txt
Si(OK=1)
   FERMER DOCUMENT($vhRefDoc)
   DOCUMENT VERS BLOB(Document;$vbCrypté)
   DOCUMENT VERS BLOB("cléPublique.txt";$vbcléPublique)
   Si(OK=1)
      DECRYPTER BLOB($vbCrypté;$vbcléPublique)
      BLOB VERS VARIABLE($vbCrypté;$vtDécrypté)
      CREER ENREGISTREMENT([Confidentiel])
      [Confidentiel]Info:=$vtDécrypté
      STOCKER ENREGISTREMENT([Confidentiel])
   Fin de si
Fin de si
```

• Utilisation de deux clés

Une société souhaite utiliser un système d'échange de données via Internet dans lequel chaque filiale reçoit des informations confidentielles mais envoie également ses propres informations à la maison-mère. Ce système a donc les impératifs suivants :
- Seul le destinataire doit pouvoir lire un message,
- On doit avoir la garantie que le message provient bien de l'expéditeur.

1. La maison-mère ainsi que chaque filiale génèrent leurs propres paires de clés (à l'aide de la méthode GENERE_CLES_TXT).

```
 `Méthode GENERE_CLES_TXT
C_BLOB($BcléPublique;$BcléPrivée)
GENERER CLES CRYPTAGE($BcléPrivée;$BcléPublique)
BLOB VERS DOCUMENT("cléPublique.txt";$BcléPublique)
BLOB VERS DOCUMENT("cléPrivée.txt";$BcléPrivée)
```

2. Chacune garde sa clé privée. Chaque filiale envoie sa clé publique à la maison-mère, qui elle-même envoie sa clé publique à chaque filiale. Cette transmission ne doit pas nécessairement être effectuée par un canal protégé, car la seule détention de la clé publique dans ce cas sera insuffisante pour décrypter une information.

3. Pour crypter une information à envoyer, une filiale ou la maison-mère exécute la méthode CRYPTER_INFOS_2 qui utilise la clé privée de l'émetteur et la clé publique du destinataire pour crypter les données :

```
 `Méthode CRYPTER_INFOS_2
C_BLOB($vbCrypté;$vbcléPrivée;$vbcléPublique)
C_TEXTE($vtCrypter)
C_HEURE($vhRefDoc)

$vtCrypter:=[Confidentiel]Info
VARIABLE VERS BLOB($vtCrypter;$vbCrypté)
 ` On charge sa propre clé privée...
DOCUMENT VERS BLOB("cléPrivée.txt";$vbcléPrivée)
Si(OK=1)
 ` ...et la clé publique du récepteur
   ALERTE("Veuillez sélectionner la clé publique du destinataire.")
   $vhRefDoc:=Ouvrir document("") `Sélection de la clé publique à charger
   Si(OK=1)
      FERMER DOCUMENT($vhRefDoc)
      DOCUMENT VERS BLOB(Document;$vbcléPublique)
 `Cryptage du BLOB avec les deux clés en paramètres
      CRYPTER BLOB($vbCrypté;$vbcléPrivée;$vbcléPublique)
      BLOB VERS DOCUMENT("MiseAJour.txt";$vbCrypté)
   Fin de si
Fin de si
```

4. Le fichier crypté peut alors être envoyé au destinataire via Internet. Si un tiers l'intercepte, il sera dans l'incapacité de le décrypter, même en connaissant les clés publiques, car il lui manquera la clé privée du destinataire.

5. Chaque destinataire peut, quant à lui, décrypter le document reçu, en utilisant sa clé privée et la clé publique de l'émetteur :

```
 `Méthode DECRYPTER_INFOS_2
C_BLOB($vbCrypté;$vbcléPublique;$vbcléPrivée)
```

```
C_TEXTE($vtDécrypté)
C_HEURE($vhRefDoc)

ALERTE("Veuillez sélectionner le document crypté.")
$vhRefDoc:=Ouvrir document("") `Sélection du fichier MiseAJour.txt
Si(OK=1)
   FERMER DOCUMENT($vhRefDoc)
   DOCUMENT VERS BLOB(Document;$vbCrypté)
 `On charge sa propre clé privée
   DOCUMENT VERS BLOB("cléPrivée.txt";$vbcléPrivée)
   Si(OK=1)
 ` ...et la clé publique de l'émetteur
      ALERTE("Veuillez sélectionner la clé publique de l'envoyeur.")
      $vhRefDoc:=Ouvrir document("") `Sélection de la clé publique
      Si(OK=1)
         FERMER DOCUMENT($vhRefDoc)
         DOCUMENT VERS BLOB(Document;$vbcléPublique)
 `Décryptage du BLOB avec les deux clés en paramètres
         DECRYPTER BLOB($vbCrypté;$vbcléPublique;$vbcléPrivée)
         BLOB VERS VARIABLE($vbCrypté;$vtDécrypté)
         CREER ENREGISTREMENT([Confidentiel])
         [Confidentiel]Info:=$vtDécrypté
         STOCKER ENREGISTREMENT([Confidentiel])
      Fin de si
   Fin de si
Fin de si
```

DECOMPRESSER BLOB

DECOMPRESSER BLOB (blob)

Paramètre	Type		Description
blob	BLOB	⇨	BLOB à décompresser

Description

DECOMPRESSER BLOB décompresse le BLOB *blob* préalablement compressé à l'aide de la commande **COMPRESSER BLOB**.

Après l'appel de la commande, la variable système OK prend la valeur 1 si le BLOB a été correctement décompressé.

Si la décompression n'a pas pu être effectuée, OK prend la valeur 0.
Dans ce cas, si l'erreur provient du fait que la mémoire disponible est insuffisante pour effectuer l'opération, aucune erreur n'est générée et la méthode poursuit son exécution.
En revanche, si l'erreur est causée par un problème plus important (le BLOB n'avait pas été compressé ou le BLOB est endommagé), l'erreur -10600 est générée. Cette erreur peut être interceptée à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

De manière générale, il est préférable d'appeler la commande **LIRE PROPRIETES BLOB** pour savoir si le BLOB a été compressé avant d'exécuter **DECOMPRESSER BLOB**.

Exemple 1

L'exemple suivant teste si le BLOB *vxMonBlob* est compressé et, si oui, le décompresse :

```
LIRE PROPRIETES BLOB(vxMonBlob;$vlCompressé;$vlTailleDécompressée;$vlTailleCourante)
Si($vlCompressé#Non compressé)
   DECOMPRESSER BLOB(vxMonBlob)
Fin de si
```

Exemple 2

L'exemple suivant vous permet de sélectionner un document et puis de le décompresser s'il était compressé :

```
$vhDocRef :=Ouvrir document("")
Si(OK=1)
   FERMER DOCUMENT($vhDocRef)
   DOCUMENT VERS BLOB(Document;vxBlob)
   Si(OK=1)
      LIRE PROPRIETES BLOB(vxBlob;$vlCompressé;$vlTailleDécompressée;$vlTailleCourante)
      Si($vlCompressé#Non compressé)
         DECOMPRESSER BLOB(vxBlob)
         Si(OK=1)
            BLOB VERS DOCUMENT(Document;vxBlob)
         Fin de si
      Fin de si
   Fin de si
Fin de si
```

Variables et ensembles système

La variable OK prend la valeur 1 si le BLOB a été correctement décompressé, sinon elle prend la valeur 0.

DECRYPTER BLOB

DECRYPTER BLOB (aDécrypter ; cléPubEmetteur {; cléPrivRécepteur})

Paramètre	Type		Description
aDécrypter	BLOB	⇨	Données à décrypter
		⇦	Données décryptées
cléPubEmetteur	BLOB	⇨	Clé publique de l'émetteur
cléPrivRécepteur	BLOB	⇨	Clé privée du récepteur

Description

La commande **DECRYPTER BLOB** permet de décrypter le contenu du BLOB *aDécrypter* à l'aide de la clé publique de l'émetteur *cléPubEmetteur* ainsi que, optionnellement, de la clé privée du récepteur *cléPrivRécepteur*.

Vous passez dans le paramètre *cléPubEmetteur* le BLOB contenant la clé publique de l'émetteur. Cette clé a été générée par l'émetteur (à l'aide de la commande **GENERER CLES CRYPTAGE**), qu'il doit ensuite transmettre au récepteur.

Le paramètre optionnel *cléPrivRécepteur* doit recevoir la clé privée du récepteur. Dans ce cas, le récepteur doit également avoir généré une paire de clés de cryptage à l'aide de **GENERER CLES CRYPTAGE** et transmis sa clé publique à l'émetteur. Le système de cryptage à deux clés permet de garantir que seul l'émetteur peut avoir crypté le message et seul le récepteur peut le décrypter.
Pour plus d'informations sur le système de cryptage à deux clés, reportez-vous à la description de la commande **CRYPTER BLOB**.

La commande **DECRYPTER BLOB** comporte une fonction de vérification d'intégrité (checksum), afin d'empêcher toute modification malveillante ou accidentelle du contenu du BLOB. Si le BLOB crypté est endommagé ou modifié, la commande ne fera rien et retournera une erreur.

Exemple

Reportez-vous aux exemples de la commande **CRYPTER BLOB**.

DOCUMENT VERS BLOB

DOCUMENT VERS BLOB (document ; blob {; *})

Paramètre	Type		Description
document	Chaîne	⇒	Nom du document
blob	BLOB	⇒	Champ ou variable de type BLOB devant recevoir le document
		⇐	Contenu du document
*	Opérateur	⇒	Macintosh seulement : la resource fork est chargée si * est passé ; sinon, la data fork est chargée

Description

DOCUMENT VERS BLOB charge le contenu de *document* dans *blob*. Vous devez passer un nom de document valide, c'est-à-dire qui désigne un document existant qui n'est pas déjà ouvert, sinon une erreur sera générée. Si vous voulez que l'utilisateur choisisse le document, utilisez la routine **Ouvrir document** et la variable système *Document* (cf. l'exemple ci-dessous).

Note pour les utilisateurs Mac OS : les documents Macintosh peuvent être composés de deux éléments, la resource fork et la data fork ("partie des ressources" et "partie des données"). Par défaut, la commande **DOCUMENT VERS BLOB** charge la data fork du document. Si vous voulez charger la resource fork du document, passez le paramètre optionnel *.
Sous Windows, le paramètre * est ignoré. Notez que l'environnement 4D vous fournit l'équivalent des resource forks de Mac OS sous Windows : par exemple, la data fork d'une base 4D est stockée dans un fichier comportant l'extension .4DB et la resource fork est stockée dans un fichier du même nom, mais comportant l'extension .RSR. Si vous développez une application 4D qui gère les data forks et les resource forks stockées dans des BLOBs, sous Windows, il vous suffit d'accéder au fichier correspondant à la "fork" avec laquelle vous voulez travailler.

Exemple

Notre exemple est une base qui vous permet de stocker et chercher rapidement des documents. Dans un formulaire entrée, vous créez un bouton qui vous permet de charger un document de votre choix dans un champ de type BLOB. La méthode pour ce bouton peut être la suivante :

```
$vhRefDoc:=Ouvrir document("") ` Sélectionner un document
Si(OK=1) ` Si un document a été choisi
   FERMER DOCUMENT($vhRefDoc) ` Nous voulons pas qu'il reste ouvert
   DOCUMENT VERS BLOB(Document;[VotreTable]VotreChampBLOB) ` Charger le document
   Si(OK=0)
  ` Gérer l'erreur
   Fin de si
Fin de si
```

Variables et ensembles système

La variable système OK prend la valeur 1 si le document est correctement lu. Sinon, elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

- Si vous essayez de charger dans un BLOB un document qui n'existe pas ou qui est déjà ouvert par un(e) autre process ou application, une des *Erreurs du gestionnaire de fichiers du système* sera générée.
- Une erreur d'Entrée/Sortie peut survenir si le document est verrouillé, s'il se trouve sur un volume verrouillé ou si un problème survient pendant la lecture du document.
- S'il n'y a pas assez de mémoire pour charger le document, une erreur -108 est générée.

Dans tous les cas, vous pouvez gérer les erreurs en utilisant la commande **APPELER SUR ERREUR**.

ENTIER LONG VERS BLOB

ENTIER LONG VERS BLOB (entierLong ; blob ; ordreOctet {; offset | *})

Paramètre	Type		Description
entierLong	Entier long	⇒	Valeur de type Entier long à écrire dans BLOB
blob	BLOB	⇒	BLOB devant recevoir l'entier long
ordreOctet	Entier long	⇒	0=Ordre d'octets natif, 1=Ordre d'octets Macintosh, 2=Ordre d'octets PC
offset | *	Variable, Opérateur	⇒	Offset (en octets) dans le BLOB ou * pour ajouter la valeur à la fin du BLOB
		⇐	Nouvel offset après l'écriture si * omis

Description

La commande **ENTIER LONG VERS BLOB** écrit la valeur de type Entier long (4 octets) *entierLong* dans le BLOB *blob*.

Le paramètre *ordreOctet* fixe l'ordre des octets ("byte ordering") de la valeur Entier long à écrire. Vous passez une des constantes fournies par 4D :

Constante	Type	Valeur
Ordre octets Macintosh	Entier long	1
Ordre octets natif	Entier long	0
Ordre octets PC	Entier long	2

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous passez le paramètre optionnel *, l'entier long sur 4 octets est ajouté à la fin du BLOB et la taille de *blob* est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de valeurs de type Entier, Entier long, Numérique ou Texte (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre *offset*, l'entier long est stocké au début de *blob* en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Si vous passez une variable dans le paramètre *offset*, l'entier long est écrit à partir de l'offset *offset*, exprimé en octets (à partir de zéro), du BLOB. Quel que soit l'endroit où vous placez l'entier long, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à 4 octets le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre *offset* est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre valeur juste après celle que vous venez d'écrire.

Exemple 1

Après l'exécution de ce code :

```
ENTIER LONG VERS BLOB(0x01020304;vxBlob;Ordre octets natif)
```

- La taille de *vxBlob* est 4 octets
- Sur plate-forme PowerPC *vxBLOB{0}=$01*, *vxBLOB{1}=$02*, *vxBLOB{2}=$03*, *vxBLOB{3}=$04*
- Sur plate-forme Intel *vxBLOB{0}=$04*, *vxBLOB{1}=$03*, *vxBLOB{2}=$02*, *vxBLOB{3}=$01*

Exemple 2

Après l'exécution de ce code :

```
ENTIER LONG VERS BLOB(0x01020304;vxBlob;Ordre octets Macintosh)
```

- La taille de *vxBlob* est 4 octets
- Sur toutes les plates-formes *vxBLOB{0}=$01*, *vxBLOB{1}=$02*, *vxBLOB{2}=$03*, *vxBLOB{3}=$04*

Exemple 3

Après l'exécution de ce code :

```
ENTIER LONG VERS BLOB(0x01020304;vxBlob;Ordre octets PC)
```

- La taille de *vxBlob* est 4 octets
- Sur toutes les plates-formes *vxBLOB{0}=$04*, *vxBLOB{1}=$03*, *vxBLOB{2}=$02*, *vxBLOB{3}=$01*

Exemple 4

Après l'exécution de ce code :

```
FIXER TAILLE BLOB(vxBlob;100)
ENTIER LONG VERS BLOB(0x01020304;vxBlob;Ordre octets PC)
```

- La taille de *vxBlob* est 104 octets

- Sur toutes les plates-formes *vxBLOB{100}=$04*, *vxBLOB{101}=$03*, *vxBLOB{102}=$02*, *vxBLOB{103}=$01*
- Les autres octets du BLOB sont inchangés

Exemple 5

Après l'exécution de ce code :

```
FIXER TAILLE BLOB(vxBlob;100)
vlOffset:=50
ENTIER LONG VERS BLOB(0x01020304;vxBlob;Ordre octets Macintosh;vlOffset)
```

- La taille de *vxBlob* est 100 K
- Sur toutes les plates-formes *vxBLOB{50}=$01*, *vxBLOB{51}=$02*, *vxBLOB{52}=$03*, *vxBLOB{53}=$04*
- Les autres octets du BLOB restent inchangés
- La variable *vlOffset* a été incrémentée de 4 (et est alors égale à 54)

ENTIER VERS BLOB

ENTIER VERS BLOB (entier ; blob ; ordreOctet {; offset | *})

Paramètre	Type		Description
entier	Entier long	⇒	Valeur entière à écrire dans le BLOB
blob	BLOB	⇒	BLOB devant recevoir la valeur entière
ordreOctet	Entier long	⇒	0=Ordre des octets en mode natif, 1=Ordre des octets Macintosh, 2=Ordre des octets PC
offset | *	Variable, Opérateur	⇒	Offset (en octets) de l'entier dans le BLOB ou * pour ajouter la valeur à la fin du BLOB
		⇐	Nouvel offset après écriture si * omis

Description

ENTIER VERS BLOB écrit la valeur entière (2 octets) *entier* dans le BLOB *blob*.

Le paramètre *ordreOctet* fixe l'ordre des octets ("byte ordering") de la valeur entière à écrire. Vous pouvez passer une des constantes fournies par 4D :

Constante	Type	Valeur
Ordre octets Macintosh	Entier long	1
Ordre octets natif	Entier long	0
Ordre octets PC	Entier long	2

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette commande.

Si vous passez le paramètre optionnel *, la valeur entière sur 2 octets est ajoutée à la fin du BLOB et sa taille est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de valeurs de type Entier, Entier long, Numérique ou Texte (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre *offset*, la valeur entière est stockée au début de *blob* en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Si vous passez une variable dans le paramètre *offset*, la valeur entière est écrite à partir de l'offset *offset*, exprimé en octets (à partir de zéro), du BLOB. Quel que soit l'endroit où vous placez l'entier, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à 2 octets le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre *offset* est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre valeur juste après celle que vous venez d'écrire.

Exemple 1

Après l'exécution de ce code :

```
ENTIER VERS BLOB(0x0206;vxBlob;Ordre octets natif)
```

- La taille de *vxBlob* est 2 octets
- Sur plate-forme PowerPC *vxBLOB{0}* = *$02* et *vxBLOB{1}* = *$06*
- Sur plate-forme Intel *vxBLOB{0}* = *$06* et *vxBLOB{1}* = *$02*

Exemple 2

Après l'exécution de ce code :

```
ENTIER VERS BLOB(0x0206;vxBlob;Ordre octets Macintosh)
```

- La taille de *vxBlob* est 2 octets
- Sur toutes les plates-formes *vxBLOB{0}* = *$02* et *vxBLOB{1}* = *$06*

Exemple 3

Après l'exécution de ce code :

```
ENTIER VERS BLOB(0x0206;vxBlob;Ordre octets PC)
```

- La taille de *vxBlob* est 2 octets
- Sur toutes les plates-formes *vxBLOB{0}* = *$06* et *vxBLOB{1}* = *$02*

Exemple 4

Après l'exécution de ce code:

```
FIXER TAILLE BLOB(vxBlob;100)
ENTIER VERS BLOB(0x0206;vxBlob;Ordre octets PC;*)
```

- La taille de *vxBlob* est 102 octets
- Sur toutes les plates-formes *vxBLOB{100}* = *$06* et *vxBLOB{101}* = *$02*

- Les autres octets du BLOB restent inchangés

Exemple 5

Après l'exécution de ce code :

```
FIXER TAILLE BLOB(vxBlob;100)
vlOffset:=50
ENTIER VERS BLOB(518;vxBlob;Ordre octets Macintosh;vlOffset)
```

- La taille de *vxBlob* est 100 octets
- Sur toutes les plates-formes *vxBLOB{50}* = *$02* et *vxBLOB{51}* = *$06*
- Les autres octets du BLOB restent inchangés
- La variable *vlOffset* est incrémentée de 2 (et est alors égale à 52)

FIXER TAILLE BLOB

FIXER TAILLE BLOB (blob ; taille {; remplisseur})

Paramètre	Type		Description
blob	BLOB	⇒	Champ ou variable de type BLOB
taille	Entier long	⇒	Nouvelle taille de BLOB
remplisseur	Entier long	⇒	Code du caractère de remplissage

Description

FIXER TAILLE BLOB redimensionne *blob* selon la valeur passée dans le paramètre *taille*.

Si vous souhaitez que les nouveaux octets réservés (s'il y en a) pour le BLOB soient initialisés avec une valeur particulière, passez cette valeur (comprise entre 0 et 255) dans le paramètre optionnel *remplisseur*.

Exemple 1

Lorsque vous n'avez plus besoin d'un BLOB process ou interprocess, il est préférable de libérer la mémoire qu'il occupe. Pour cela, écrivez le code suivant :

```
FIXER TAILLE BLOB(vProcessBLOB;0)
FIXER TAILLE BLOB(◊vInterprocessBLOB;0)
```

Exemple 2

L'exemple suivant crée un BLOB de 16 Ko et remplit chaque octet avec la valeur 0xFF :

```
C_BLOB(vxData)
FIXER TAILLE BLOB(vxData;16*1024;0xFF)
```

Gestion des erreurs

Si vous ne pouvez pas redimensionner le BLOB parce qu'il n'y a pas assez de mémoire, l'erreur *–108* est générée. Vous pouvez installer une méthode avec la commande **APPELER SUR ERREUR** pour interrompre la méthode lorsqu'une erreur survient.

INSERER DANS BLOB

INSERER DANS BLOB (blob ; décalage ; nombre {; remplisseur})

Paramètre	Type		Description
blob	BLOB	⇒	BLOB dans lequel insérer les octets
décalage	Entier long	⇒	Position de début d'insertion des octets
nombre	Entier long	⇒	Nombre d'octets à insérer
remplisseur	Entier long	⇒	Valeur d'octet par défaut (0x00..0xFF) 0x00 si ce paramètre est omis

Description

INSERER DANS BLOB insère le nombre d'octets spécifié par *nombre* dans le BLOB *blob* à la position spécifiée par *décalage*. La taille du BLOB est augmentée de *nombre* d'octets.

Si vous ne passez pas le paramètre optionnel *remplisseur*, la valeur des octets insérés dans le BLOB est fixée à *0x00*. Sinon, les octets prennent la valeur passée dans *remplisseur* (modulo 256 — 0..255).

Vous passez dans le paramètre *décalage* la position (relative à l'origine du BLOB) de l'insertion.

LIRE PROPRIETES BLOB

LIRE PROPRIETES BLOB (blob ; compressé {; tailleDécompressée {; tailleCourante}})

Paramètre	Type		Description
blob	BLOB	⇒	BLOB sur lequel vous voulez obtenir des informations
compressé	Entier long	⇐	0 = pas de compression, 1 = interne compact, 2 = interne rapide, -1 = GZIP compact, -2 = GZIP rapide
tailleDécompressée	Entier long	⇐	Taille du BLOB décompressé en octets
tailleCourante	Entier long	⇐	Taille courante du BLOB en octets

Description

LIRE PROPRIETES BLOB retourne des informations sur le BLOB *blob*.

Le paramètre *compressé* retourne une valeur indiquant si et comment le BLOB est compressé. Vous pouvez comparer cette valeur aux constantes suivantes, placées dans le thème **BLOB** :

Constante	Type	Valeur	Comment
GZIP méthode de compression compacte	Entier long	-1	Compression GZIP la plus compacte (au détriment de la vitesse à laquelle la compression et la décompression sont effectuées)
GZIP méthode de compression rapide	Entier long	-2	Compression/décompression GZIP la plus rapide (au détriment du taux de compression)
Méthode de compression compacte	Entier long	1	Compression interne la plus compacte (au détriment de la vitesse à laquelle la compression et la décompression sont effectuées). Méthode par défaut.
Méthode de compression rapide	Entier long	2	Compression/décompression interne la plus rapide au détriment du taux de compression (une fois compressé, le BLOB prend plus de place)
Non compressé	Entier long	0	Pas de compression

Quel que soit l'état de compression du BLOB, le paramètre *tailleDécompressée* retourne la taille du BLOB non compressé.

Le paramètre *tailleCourante* retourne la taille courante du BLOB. Si le BLOB est compressé, *tailleCourante* sera inférieur à *tailleDécompressée*. Si le BLOB n'est pas compressé, *tailleCourante* sera égal à *tailleDécompressée*.

Exemple 1

Référez-vous aux exemples des commandes **COMPRESSER BLOB** et **DECOMPRESSER BLOB**.

Exemple 2

Lorsqu'un BLOB est compressé, la méthode projet suivante vous permet de connaître le taux de place gagnée en compressant le BLOB :

```
   ` Méthode projet Place gagnée par compression
   ` Place gagnée par compression (Pointeur {; Pointeur } ) -> Entier long
   ` Place gagnée par compression ( -> BLOB {; -> octetsGagnés } ) -> Pourcentage

C_POINTEUR($1;$2)
C_ENTIER LONG($0;$vlCompressé;$vlTailleDécompressée;$vlTailleCourante)

LIRE PROPRIETES BLOB($1->;$vlCompressé;$vlTailleDécompressée;$vlTailleCourante)
Si($vlTailleDécompressée=0)
   $0:=0
   Si(Nombre de paramètres>=2)
      $2->:=0
   Fin de si
Sinon
   $0:=100-(($vlTailleCourante/$vlTailleDécompressée)*100)
   Si(Nombre de paramètres>=2)
      $2->:=$vlTailleDécompressée-$vlTailleCourante
   Fin de si
Fin de si
```

Lorsque cette méthode est placée dans votre application, vous pouvez écrire :

```
   ` ...
COMPRESSER BLOB(vxBlob)
$vlPourcent:=Place gagnée par compression(->vxBlob;->vlTailleBlob)
ALERTE("La compression permet de gagner "+Chaine(vlTailleBlob)+" octets, donc "+Chaine($vlPourcent;"#0%")+"
d'espace.")
```

LISTE VERS BLOB

LISTE VERS BLOB (liste ; blob {; *})

Paramètre	Type		Description
liste	RefListe	⇨	Liste hiérarchique à stocker dans le BLOB
blob	BLOB	⇨	BLOB devant recevoir la liste hiérarchique
*	Opérateur	⇨	Ajouter la liste à la fin du BLOB

Description

La commande **LISTE VERS BLOB** stocke la liste hiérarchique *liste* dans le BLOB *blob*.

Si vous passez le paramètre optionnel *, la liste hiérarchique est ajoutée à la fin du BLOB et la taille de *blob* est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de variables ou de listes (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel *, la liste hiérarchique est stockée au début de *blob* en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Quel que soit l'endroit où vous placez la liste, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à la taille de la liste le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

ATTENTION : Si vous utilisez un BLOB pour stocker des listes, appelez ensuite la commande **BLOB vers liste** pour relire le contenu du BLOB car les listes sont stockées dans les BLOBs avec un format interne 4D.

Après l'exécution de la commande, la variable OK prend la valeur 1 si la liste hiérarchique a été correctement stockée. Si l'opération n'a pas pu être effectuée car, par exemple, il n'y avait pas assez de mémoire disponible, la variable OK prend la valeur 0.

Note pour l'indépendance de plate-forme : LISTE VERS BLOB et **BLOB vers liste** utilisent un format interne 4D pour gérer les listes stockées dans des BLOBs. L'avantage est que vous n'avez pas besoin de vous soucier de la conversion des octets ("byte swapping") entre les plates-formes lorsque vous utilisez ces deux commandes. Autrement dit, avec ces commandes, un BLOB créé sous Windows peut être réutilisé sous Mac OS et vice-versa.

Exemple

Reportez-vous à l'exemple de la fonction **BLOB vers liste**.

REEL VERS BLOB

REEL VERS BLOB (réel ; blob ; formatRéel {; offset | *})

Paramètre	Type		Description
réel	Réel	➡	Valeur de type Réel à écrire dans le BLOB
blob	BLOB	➡	BLOB devant recevoir la valeur Réel
formatRéel	Entier long	➡	0=Format réel natif, 1=Format réel étendu, 2=Format réel double Macintosh, 3=Format réel double Windows
offset | *	Variable, Opérateur	➡	Offset (en octets) dans le BLOB ou * pour ajouter la valeur à la fin du BLOB
		⬅	Nouvel offset après l'écriture si * omis

Description

La commande **REEL VERS BLOB** écrit la valeur de type Réel *réel* dans le BLOB *blob*.

Le paramètre *formatRéel* fixe le format interne et l'ordre des octets ("byte ordering") de la valeur de type Réel à écrire. Vous passez une des constantes fournies par 4D :

Constante	Type	Valeur
Format réel double Macintosh	Entier long	2
Format réel double PC	Entier long	3
Format réel étendu	Entier long	1
Format réel natif	Entier long	0

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous passez le paramètre optionnel *, la valeur réelle est ajoutée à la fin du BLOB et la taille de *blob* est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de valeurs de type Entier, Entier long, Numérique ou Texte (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre *offset*, la valeur réelle est stockée au début de *blob* en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Si vous passez une variable dans le paramètre *offset*, le réel est écrit à partir de l'offset *offset*, exprimé en octets (à partir de zéro), du BLOB. Quel que soit l'endroit où vous placez la valeur, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à 8 ou 10 octets le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre *offset* est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre valeur juste après celle que vous venez d'écrire.

Exemple 1

Après l'exécution de ce code :

```
C_REEL(vrValeur)
vrValeur:=...
REEL VERS BLOB(vrValeur;vxBlob;Format réel natif)
```

- Sur toutes les plates-formes, la taille de *vxBlob* est 8 octets

Exemple 2

Après l'exécution de ce code :

```
C_REEL(vrValeur)
vrValeur:=...
REEL VERS BLOB(vrValeur;vxBlob;Format réel étendu)
```

- Sur toutes les plates-formes, la taille de *vxBlob* est 10 octets

Exemple 3

Après l'exécution de ce code :

```
C_REEL(vrValeur)
vrValeur:=...
REEL VERS BLOB(vrValeur;vxBlob;Format réel double Macintosh) ` ou Format réel double PC
```

- Sur toutes les plates-formes, la taille de *vxBlob* est 8 octets

Exemple 4

Après l'exécution de ce code :

```
FIXER TAILLE BLOB(vxBlob;100)
```

```
C_REEL(vrValeur)
vrValeur:=...
REEL VERS BLOB(vrValeur;vxBlob;Format réel double PC) ` ou Format réel double Macintosh
```

- Sur toutes les plates-formes, la taille de *vxBlob* est 8 octets

Exemple 5

Après l'exécution de ce code :

```
FIXER TAILLE BLOB(vxBlob;100)
REEL VERS BLOB(vrValeur;vxBlob;Format réel étendu;*)
```

- Sur toutes les plates-formes, la taille de *vxBlob* est 110 octets
- Sur toutes les plates-formes, la valeur numérique est stockée dans les octets #100 à #109
- Les autres octets du BLOB restent inchangés

Exemple 6

Après l'exécution de ce code :

```
FIXER TAILLE BLOB(vxBlob;100)
C_REEL(vrValeur)
vrValeur:=...
vlOffset:=50
REEL VERS BLOB(vrValeur;vxBlob;Format réel double PC;vlOffset) ` ou Format réel double Macintosh
```

- Sur toutes les plates-formes, la taille de *vxBlob* est 100 octets
- Sur toutes les plates-formes, la valeur numérique est stockée dans les octets #50 à #57
- Les autres octets du BLOB restent inchangés
- La variable *vlOffset* est incrémentée de 8 (et est alors égale à 58)

SUPPRIMER DANS BLOB

SUPPRIMER DANS BLOB (blob ; offset ; nombre)

Paramètre	Type		Description
blob	BLOB	⇒	BLOB duquel supprimer des octets
offset	Entier long	⇒	Offset à partir duquel supprimer les octets
nombre	Entier long	⇒	Nombre d'octets à supprimer

Description

SUPPRIMER DANS BLOB supprime le nombre d'octets spécifié par *nombre* du BLOB *blob* à partir de la position définie par *offset* (exprimée de manière relative à l'origine du BLOB). La taille du BLOB est réduite de *nombre* d'octets.

Taille BLOB

Taille BLOB (blob) -> Résultat

Paramètre	Type		Description
blob	BLOB	→	Champ ou variable de type BLOB
Résultat	Entier long	↩	Taille en octets du BLOB

Description

Taille BLOB retourne la taille de *blob* exprimée en octets.

Exemple

La ligne de code suivante ajoute 100 octets au BLOB *monBlob* :

```
FIXER TAILLE BLOB(Taille BLOB(monBlob)+100)
```

TEXTE VERS BLOB

TEXTE VERS BLOB (texte ; blob {; formatTexte {; offset | *}})

Paramètre	Type		Description
texte	Chaîne	⇒	Texte à écrire dans blob
blob	BLOB	⇒	BLOB devant recevoir le texte
formatTexte	Entier long	⇒	Format et jeu de caractères du texte
offset | *	Variable, Opérateur	⇒	Offset (en octets) dans le BLOB ou * pour ajouter la valeur à la fin du BLOB
		⇐	Nouvel offset après l'écriture si * omis

Description

La commande **TEXTE VERS BLOB** écrit la valeur de type Texte *texte* dans le BLOB *blob*.

Le paramètre *formatTexte* permet de définir le format interne et le jeu de caractères de la valeur de type Texte à écrire. Pour cela, passez dans *formatTexte* une des constantes suivantes, placées dans le thème "**BLOB**" :

Constante	Type	Valeur
Mac chaîne en C	Entier long	0
Mac chaîne pascal	Entier long	1
Mac texte avec longueur	Entier long	2
Mac texte sans longueur	Entier long	3
UTF8 chaîne en C	Entier long	4
UTF8 texte avec longueur	Entier long	5
UTF8 texte sans longueur	Entier long	6

Si vous omettez le paramètre *formatTexte*, par défaut 4D utilise le format Mac chaîne en C. Dans les bases de données créées à compter de la version 11, 4D travaille par défaut avec le jeu de caractères Unicode (UTF8) pour la gestion des textes, il est donc recommandé d'utiliser ce jeu de caractères.

Notes :

- Les constantes "UTF8" sont utilisables uniquement lorsque l'application fonctionne en mode Unicode.
- Les constantes "Mac" ne permettent pas de traiter des textes de plus de 32 ko.
- Si vous souhaitez manipuler des jeux de caractères autres que UTF8, utilisez la commande **CONVERTIR DEPUIS TEXTE**.

Le tableau suivant décrit chacun de ces formats :

Format texte	Description et Exemples	
Chaîne en C	Le texte se termine par un caractère NULL (code ASCII $00).	
	UTF8	"" --> $00
		"Café" --> $43 61 66 C3 A9 00
	Mac	"" --> $00
		"Café" --> $43 61 66 8E 00
Chaîne pascal	Le texte est précédé d'un octet de longueur.	
	UTF8	-
		-
	Mac	"" --> $00
		"Café" --> $04 43 61 66 8E
Texte avec longueur	Le texte est précédé de 4 octets (UTF8) ou 2 octets (Mac) de longueur.	
	UTF8	"" --> $00 00 00 00
		"Café" --> $00 00 00 05 43 61 66 C3 A9
	Mac	"" --> $00 00
		"Café" --> $00 04 43 61 66 8E
Texte sans longueur	Le texte est composé seulement de ses caractères.	
	UTF8	"" --> Pas de valeur
		"Café" --> $43 61 66 C3 A9
	Mac	"" --> Pas de valeur
		"Café" --> $43 61 66 8E

Si vous passez le paramètre optionnel *, la valeur de type Texte est ajoutée à la fin du BLOB et la taille de *blob* est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de valeurs de type Entier, Entier long, Numérique ou Texte (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre *offset*, la valeur de type Texte est stockée au début de *blob* en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Si vous passez une variable dans le paramètre *offset*, la valeur de type Texte est écrite à l'offset *offset*, exprimé en octets (à partir de zéro), du BLOB. Quel que soit l'endroit où vous placez la valeur, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à la taille du texte le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre *offset* est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre valeur juste après celle que vous venez d'écrire.

Exemple

Après l'exécution de ce code :

```
FIXER TAILLE BLOB(vxBlob;0)
```

```
C_TEXTE(vtValeur)
vtValeur:="Café" ` La longueur de vtValeur est de 4 octets
TEXTE VERS BLOB(vtValeur;vxBlob;Mac chaîne en C) ` La taille du BLOB devient 5 octets
TEXTE VERS BLOB(vtValeur;vxBlob;Mac chaîne pascal) ` La taille du BLOB devient 5 octets
TEXTE VERS BLOB(vtValeur;vxBlob;Mac texte avec longueur) ` La taille du BLOB devient 6 octets
TEXTE VERS BLOB(vtValeur;vxBlob;Mac texte sans longueur) ` La taille du BLOB devient 4 octets
TEXTE VERS BLOB(vtValeur;vxBlob;UTF8 chaîne en C) ` La taille du BLOB devient 6 octets
TEXTE VERS BLOB(vtValeur;vxBlob;UTF8 texte avec longueur) ` La taille du BLOB devient 9 octets
TEXTE VERS BLOB(vtValeur;vxBlob;UTF8 texte sans longueur) ` La taille du BLOB devient 5 octets
```

VARIABLE VERS BLOB

VARIABLE VERS BLOB (variable ; blob {; offset | *})

Paramètre	Type		Description
variable	Variable	➡	Variable à stocker dans le BLOB
blob	BLOB	➡	BLOB devant recevoir la variable
offset | *	Variable, Opérateur	➡	Offset de la variable (en octets) dans BLOB ou * pour ajouter la variable à la fin du BLOB
		⬅	Nouvel offset après écriture si * omis

Description

VARIABLE VERS BLOB stocke la variable *variable* dans le BLOB *blob*.

Si vous passez le paramètre optionnel *, la *variable* est ajoutée à la fin de *blob* et la taille du BLOB est redimensionnée en conséquence. A l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de variables ou de listes (cf. les autres commandes BLOB) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre *offset*, *variable* est stockée à partir du début du BLOB en écrasant son contenu précédent. La taille du BLOB est redimensionnée en conséquence.

Si vous passez la variable *offset* en paramètre, la *variable* est écrite dans le BLOB à l'offset (à partir de zéro) spécifié par *offset*. Quel que soit l'endroit où vous placez la variable, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (ainsi que de la taille de la variable). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre *offset* est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre variable ou liste juste après celle que vous venez d'écrire.

VARIABLE VERS BLOB accepte tous les types de variables (y compris d'autres BLOBs), à l'exception des types suivants :

- Pointeurs
- Tableaux de pointeurs

A noter que :

- si vous stockez une variable de type Entier long qui est une référence à une liste hiérarchique (ListRef), **VARIABLE VERS BLOB** stockera la variable Entier long, pas la liste. Pour stocker et récupérer des listes hiérarchiques dans un BLOB, utilisez les commandes **LISTE VERS BLOB** et **BLOB vers liste**.
- si vous passez dans le paramètre *variable* un objet **C_OBJET**, la commande le place dans le BLOB sous la forme JSON en utf-8. Si l'objet contient des pointeurs, leur valeurs dépointées sont stockées dans le BLOB, pas les pointeurs eux-mêmes.

ATTENTION : Si vous utilisez un BLOB pour stocker les variables, utilisez par la suite la commande **BLOB VERS VARIABLE** pour récupérer le contenu du BLOB car les variables sont stockées dans les BLOBs avec un format interne à 4D.

La variable OK prend la valeur 1 si la variable a été correctement stockée. Si l'opération n'a pas pu être effectuée à cause d'un manque de mémoire, la variable OK prend la valeur 0.

Note sur l'indépendance de plate-forme : VARIABLE VERS BLOB et **BLOB VERS VARIABLE** utilisent un format interne à 4D pour gérer les variables stockées dans les BLOBs. Vous n'avez donc pas besoin de vous préoccuper de la conversion des octets ("byte swapping") entre les différentes plates-formes lors de l'utilisation de ces deux commandes. Un BLOB créé sous Windows à l'aide de ces deux commandes peut être réutilisé sans la moindre manipulation sous Mac OS et vice-versa.

Exemple 1

Les méthodes projet suivantes vous permettent de stocker et de récupérer rapidement des variables dans les documents sur disque :

```
  ` Méthode projet STOCKER VARIABLES
  ` STOCKER VARIABLES ( Alpha ; Pointeur )
  ` STOCKER VARIABLES ( Document ; -> Tableau )
C_ALPHA(255;$1)
C_POINTEUR($2)
C_BLOB($vxDonnéesTableau)
VARIABLE VERS BLOB($2->;$vxDonnéesTableau) ` Stocker le tableau dans le BLOB
COMPRESSER BLOB($vxDonnéesTableau) ` Compresser le BLOB
BLOB VERS DOCUMENT($1;$vxDonnéesTableau) ` Enregistrer le BLOB sur disque

  ` Méthode projet CHARGER VARIABLES
  ` CHARGER VARIABLES ( Alpha ; Pointeur )
  ` CHARGER VARIABLES ( Document ; -> Tableau )
C_ALPHA(255;$1)
C_POINTEUR($2)
C_BLOB($vxDonnéesTableau)
DOCUMENT VERS BLOB($1;$vxDonnéesTableau) ` Charger le BLOB du disque
DECOMPRESSER BLOB($vxDonnéesTableau) ` Décompresser le BLOB
BLOB VERS VARIABLE($vxDonnéesTableau;$2->) ` Récupérer le tableau du BLOB
```

Lorsque ces méthodes ont été ajoutées à votre application, vous pouvez écrire :

```
TABLEAU ALPHA(...;taToutTableau;...)
  ` ...
STOCKER VARIABLES($vaNomDoc;->taToutTableau)
  ` ...
CHARGER VARIABLES($vaNomDoc;->taToutTableau)
```

Exemple 2

Les deux méthodes projet suivantes vous permettent de stocker et de récupérer des variables dans un BLOB :

```
  ` Méthode projet STOCKER VARIABLES DANS BLOB
  ` STOCKER VARIABLES DANS BLOB ( Pointeur { ; Pointeur ... { ; Pointeur } } )
  ` STOCKER VARIABLES DANS BLOB ( BLOB { ; Var1 ... { ; Var2 } } )
C_POINTEUR(${1})
C_ENTIER LONG($vlParam)

FIXER TAILLE BLOB($1->;0)
Boucle($vlParam;2;Nombre de paramètres)
   VARIABLE VERS BLOB(${$vlParam}->;$1->;*)
Fin de boucle


  ` Méthode projet RECUPERER VARIABLES DANS BLOB
  ` RECUPERER VARIABLES DANS BLOB ( Pointeur { ; Pointeur ... { ; Pointeur } } )
  ` RECUPERER VARIABLES DANS BLOB ( BLOB { ; Var1 ... { ; Var2 } } )
C_POINTEUR(${1})
C_ENTIER LONG($vlParam;$vlOffset)

$vlOffset:=0
Boucle($vlParam;2;Nombre de paramètres)
   BLOB VERS VARIABLE($1->;${$vlParam}->;$vlOffset)
Fin de boucle
```

Lorsque ces méthodes ont été ajoutées à votre application, vous pouvez écrire :

```
STOCKER VARIABLES DANS BLOB(->vxBLOB;->vgImage;->taTableau1;->taTableau2)
  ` ...
RECUPERER VARIABLES DANS BLOB(->vxBLOB;->vgImage;->taTableau1;->taTableau2)
```

Variables et ensembles système

La variable OK prend la valeur 1 si la variable a été correctement stockée, sinon elle prend la valeur 0.

Booléens

- Commandes du thème Booléens
- Faux
- Non
- Vrai

Commandes du thème Booléens

Les fonctions booléennes de 4D sont utilisées pour les opérations de type booléennes, c'est-à-dire ne traitant que deux valeurs, VRAI ou FAUX. Ces fonctions sont les suivantes :

- **Vrai**
- **Faux**
- **Non**

Exemple

L'exemple suivant retourne **Vrai** dans la variable *monBooléen* si l'utilisateur a cliqué sur le bouton *monBouton* et **Faux** s'il n'a pas cliqué dessus. Lorsqu'un bouton reçoit un clic, la variable du bouton prend la valeur 1. Dans cet exemple, la valeur de la variable booléenne est basée sur la valeur d'un bouton.

```
Si(monBouton=1) ` Si le bouton a reçu un clic
   monBooléen:=Vrai ` monBooléen prend la valeur Vrai
Sinon ` Si le bouton n'a pas reçu de clic,
   monBooléen:=Faux ` monBooléen prend la valeur Faux
Fin de si
```

L'exemple ci-dessus peut être simplifié et écrit en une seule ligne :

```
monBooléen:=(monBouton=1)
```

Faux

Faux -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Faux

Description

Faux retourne la valeur booléenne Faux.

Exemple

L'exemple suivant met la variable *vbOptions* à Faux :

```
vbOptions:=Faux
```

Non

Non (booléen) -> Résultat

Paramètre	Type		Description
booléen	Booléen	⇨	Valeur booléenne à inverser
Résultat	Booléen	⮌	Inverse de booléen

Description

La fonction **Non** retourne la valeur inverse de *booléen*, changeant un **Vrai** en **Faux** ou un **Faux** en **Vrai**.

Exemple

Dans l'exemple suivant, la valeur **Vrai** est assignée à une variable. Cette valeur est alors modifiée en **Faux** puis de nouveau en **Vrai** :

```
Résultat:=Vrai ` Résultat prend la valeur VRAI
Résultat:=Non(Résultat) ` Résultat prend la valeur FAUX
Résultat:=Non(Résultat) ` Résultat prend la valeur VRAI
```

Vrai

Vrai -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Vrai

Description

Vrai retourne la valeur booléenne Vrai.

Exemple

L'exemple suivant met la variable *vbOptions* à Vrai :

```
vbOptions:=Vrai
```

Chaînes de caractères

- Symboles d'indice de chaîne
- Balises de transformation 4D
- Caractere
- Chaine
- Code de caractere
- CONVERTIR DEPUIS TEXTE
- Convertir vers texte
- Inserer chaine
- LIRE MOTS CLES TEXTE
- Lire traduction chaine
- Longueur
- Majusc
- Minusc
- Num
- Position
- Remplacer caracteres
- Remplacer chaine
- Sous chaine
- Supprimer chaine
- Trouver regex
- *_o_Convertir caractères*
- *_o_ISO vers Mac*
- *_o_Mac vers ISO*
- *_o_Mac vers Windows*
- *_o_Windows vers Mac*

Symboles d'indice de chaîne

Introduction

Les symboles d'indice de chaîne sont les suivants : **[[...]]**

Ces symboles sont utilisés pour désigner un caractère particulier dans une chaîne. Cette syntaxe vous permet de référencer un caractère dans un champ ou une variable de type Alpha ou Texte.

Note de compatibilité : A compter de 4D v13, il n'est plus possible de visualiser les anciens symboles Mac OS dans l'éditeur de méthodes (**≤...≥**).

Lorsque les symboles d'indice de chaîne sont placés à gauche de l'opérateur d'affectation (:=), un caractère est affecté à la position référencée dans la chaîne. Par exemple, en postulant que la chaîne *vsNom* n'est pas une chaîne vide, le code suivant passe le premier caractère de la chaîne *vsNom* en majuscule :

```
Si(vsNom#"")
   vsNom[[1]]:=Majusc(vsNom[[1]])
Fin de si
```

Lorsque les symboles d'indice de chaîne apparaissent dans une expression, ils retournent le caractère auquel ils font référence sous la forme d'une chaîne d'un caractère. En voici un exemple :

```
  ` L'exemple suivant teste si le dernier caractère de vtText est le caractère "@"
Si(vtText#"")
   Si(Code de caractere(Sous chaine(vtText;Longueur(vtText);1))=Arobase)
  ` ...
   Fin de si
Fin de si

  ` En utilisant la syntaxe des caractères d'indice de chaîne, vous écririez plus simplement :
Si(vtText#"")
   Si(Code de caractere(vtText[[Longueur(vtText)]]=Arobase)
  ` ...
   Fin de si
Fin de si
```

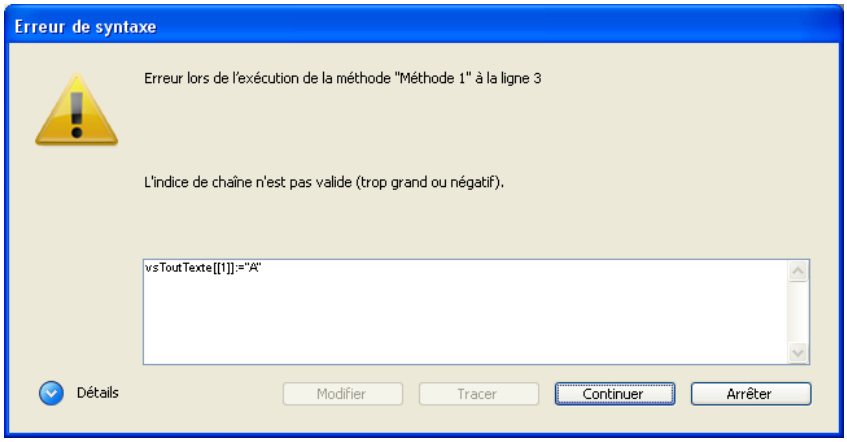
Note avancée sur la référence à des caractères invalides

Lorsque vous utilisez les symboles d'indice de chaîne, **il est de votre responsabilité** de vous référer à des caractères existant dans la chaîne, de la même manière que pour les éléments d'un tableau. Si, par exemple, vous référencez le 20e caractère d'une chaîne, cette chaîne doit contenir au moins 20 caractères.

- Ne pas respecter cette condition en mode interprété n'est pas signalé comme une erreur par 4D.
- Ne pas respecter cette condition en mode compilé (sans options) peut entraîner une "corruption" de la mémoire, si, par exemple, vous écrivez un caractère au-delà de la fin d'une chaîne ou d'un texte.
- Ne pas respecter cette condition en mode compilé est signalé lorsque le contrôle d'exécution est activé. Si, par exemple, vous exécutez le code suivant :

```
  ` Ne pas faire ça !
vsToutTexte:=""
vsToutTexte[[1]]:="A"
```

L'alerte suivante s'affichera :



Exemple

La méthode projet suivante ajoute une lettre capitale à tous les mots du texte passé en paramètre et retourne le texte modifié :

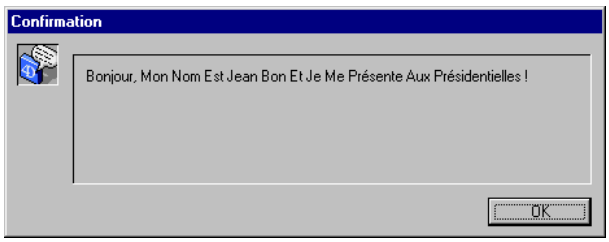
```
  ` Méthode projet de passage en capitale
```

```
   ` PasserEnCap ( Texte ) -> Texte
   ` PasserEnCap ( Texte source ) -> Texte avec des lettres capitales

$0:=$1
$vlLen:=Longueur($0)
Si($vlLen>0)
   $0[[1]]:=Majusc($0[[1]])
   Boucle($vlChar;1;$vlLen-1)
      Si(Position($0[[$vlChar]];" !&()-{}:;<>?/,.=+*")>0)
         $0[[$vlChar+1]]:=Majusc($0[[$vlChar+1]])
      Fin de si
   Fin de boucle
Fin de si
```

Une fois cette méthode placée dans la base, la ligne :

```
ALERTE(PasserEnCap("Bonjour, mon nom est Jean Bon et je me présente aux présidentielles !"))
```

... affiche l'alerte suivante :



4D propose un jeu de balises de transformation permettant d'insérer des références à des expressions ou variables 4D ou d'effectuer différents types de traitements à l'intérieur d'un texte source, appelé "template". Ces balises sont interprétées lors de l'exécution du texte source et génèrent un texte de sortie.

Ce principe est notamment utilisé par le serveur Web 4D pour construire des **Pages semi-dynamiques**.

Ces balises doivent être généralement insérées sous forme de commentaires type HTML (*<!--#LaBalise LeContenu-->*) dans le texte source. La présence d'autres commentaires (par exemple *<!--Début de liste-->*) est toutefois possible.

Note : Une syntaxe alternative utilisant le $ est utilisable dans certaines conditions pour les balises qui retournent des valeurs, afin de les rendre conformes au XML. Pour plus d'informations, reportez-vous ci-dessous au paragraphe **Syntaxe alternative pour 4DTEXT, 4DHTML, 4DEVAL**.

Les balises de transformation 4D disponibles sont les suivantes :

- 4DTEXT, pour insérer des variables et expressions 4D en tant que texte
- 4DHTML, pour insérer du code HTML
- 4DEVAL, pour évaluer toute expression 4D
- 4DSCRIPT, pour exécuter une méthode 4D
- 4DINCLUDE, pour insérer une page HTML dans une autre page
- 4DBASE, pour modifier le dossier par défaut utilisé par la balise 4DINCLUDE
- 4DCODE, pour insérer des blocs de code 4D
- 4DIF, 4DELSE, 4DELSEIF et 4DENDIF, pour insérer des conditions dans le code balisé
- 4DLOOP et 4DENDLOOP, pour insérer des boucles dans le code balisé

Il est possible d'imbriquer les différents types de balises. Voici par exemple une structure HTML parfaitement envisageable :

```
<HTML> ... <BODY> <!--#4DSCRIPT/PRE_PROCESS-->          (Appel de méthode) <!--#4DIF (mavar=1)--
>                (Si condition)    <!--#4DINCLUDE banner1.html-->   (Insertion d'une sous page) <!--#4DENDIF--
>                     (Fin de si) <!--#4DIF (mavar=2)-->    <!--#4DINCLUDE banner2.html--> <!--#4DENDIF--> <!--
#4DLOOP [TABLE]-->               (Boucle sur la sélection courante) <!--#4DIF ([TABLE]ValNum>10)-->          (Si
[TABLE]ValNum>10)    <!--#4DINCLUDE subpage.html-->   (Inclusion d'une sous page) <!--#4DELSE--
>                     (Sinon)    <B>Valeur : <!--#4DTEXT [TABLE]ValNum--></B><BR>
                              (Affichage d'un champ) <!--#4DENDIF--> <!--#4DENDLOOP-->                  (Fin de
boucle) </BODY> </HTML>
```

Exécution des templates

L'analyse du contenu des pages 'templates' peut être effectuée de deux manières :

- Utilisation de la commande **TRAITER BALISES 4D** ; cette commande accepte un 'template' en entrée, ainsi que des paramètres (optionnel) et retourne un texte résultant du traitement.
- Utilisation du serveur HTTP intégré de 4D : **Pages semi-dynamiques** envoyées via les commandes **WEB ENVOYER FICHIER** (.htm, .html, .shtm, .shtml), **WEB ENVOYER BLOB** (blob de type text/html), **WEB ENVOYER TEXTE** ou appelées via des URLs. Dans ce dernier cas, à des fins d'optimisation, les pages suffixées ".htm" et ".html" ne sont PAS analysées. Pour "forcer" l'analyse des pages HTML dans ce cas, vous devez les suffixer ".shtm" ou ".shtml" (par exemple http://www.server.com/dir/page.shtm). Pour plus d'informations sur ce point, reportez-vous à la section **Pages semi-dynamiques** dans le chapitre **Serveur Web**.

4DTEXT

Syntaxe : *<!--#4DTEXT NomVar-->* ou *<!--#4DTEXT Expression4D-->*
Syntaxe alternative : *$4DTEXT(NomVar)* ou *$4DTEXT(Expression4D)* (voir **Syntaxe alternative pour 4DTEXT, 4DHTML, 4DEVAL**)

La balise *<!--#4DTEXT NomVar-->* vous permet d'insérer une référence à une variable ou à une expression 4D retournant une valeur. Par exemple, si vous écrivez dans une page HTML :
<P>Bienvenue sur *<!--#4DTEXT vtNomSite-->*!</P>
La valeur de la variable 4D *vtNomSite* sera insérée dans la page HTML lors de son envoi. Cette valeur est insérée sous forme de texte simple, les caractères HTML spéciaux tels que "<" sont automatiquement échappés.

Il est aussi possible d'insérer des expressions 4D à l'aide de la balise *4DTEXT*. Vous pouvez par exemple insérer directement le contenu d'un champ (*<!--#4DTEXT [nomTable]nomChamp-->*), un élément de tableau (*<!--#4DTEXT tab{1}-->*) ou une méthode retournant une valeur (*<!--#4DTEXT mamethode-->*). La conversion de l'expression obéit aux mêmes règles que la conversion d'une variable. En outre, l'expression doit satisfaire aux règles de syntaxe de 4D.

Note : L'exécution d'une méthode 4D avec *4DTEXT* depuis une requête Web est soumise à la valeur de l'attribut "Disponible via les balises et URLs 4D (4DACTION...)" défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section **Sécurité des connexions**.

Pour assurer une évaluation correcte de l'expression quelle que soit la langue ou la version de 4D, il est recommandé d'utiliser la syntaxe *tokenisée* pour les éléments dont le nom peut varier au fil des versions (commandes, tables, champs, constantes). Par exemple, pour insérer la commande **Heure courante**, saisissez '**Current time:C178**'. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser des tokens dans les formules**.

En cas d'erreur d'évaluation, le texte inséré sera de la forme "*<!--#4DTEXT mavar-->* : ## erreur # code d'erreur".

Notes :

- Vous devez utiliser des variables process.
- Pour des raisons de sécurité, il est recommandé d'utiliser cette balise lorsque vous traitez des données introduites depuis l'extérieur de l'application, afin d'empêcher l'injection de code malveillant (cf. paragraphe **Notes d'utilisation** ci-dessous).
- Il est possible d'afficher le contenu d'un champ image. En revanche, il n'est pas possible d'afficher le contenu d'un élément de tableau image.
- Il est possible d'afficher le contenu d'un champ objet, par l'intermédiaire d'une formule 4D. Par exemple, vous pouvez écrire *<!--#4DTEXT **OB***

Lire:C1224([Rect]Desc;\"color\")-->.

- Vous travaillerez généralement avec des variables Texte. Toutefois, vous pouvez également utiliser des variables BLOB. Il vous suffit pour cela de générer des BLOBs en mode Texte sans longueur.

4DHTML

Syntaxe : *<!--#4DHTML NomVar-->* ou *<!--#4DHTML Expression4D-->*
Syntaxe alternative : *$4DHTML(NomVar)* ou *$4DHTML(Expression4D)* (voir **Syntaxe alternative pour 4DTEXT, 4DHTML, 4DEVAL**)

Tout comme la balise 4DTEXT, cette balise permet d'évaluer une variable ou une expression 4D retournant une valeur et de l'insérer en tant qu'expression HTML. A la différence de la balise 4DTEXT, cette balise n'échappe pas les caractères HTML spéciaux tels que ">".

Par exemple, voici les résultats du traitement de la variable Texte 4D *mavar* à l'aide des balises 4DTEXT et 4DHTML :

Valeur de *mavar*	Balises	Résultat
mavar:="<B>"	*<!--#4DTEXT mavar-->*	<B>
mavar:="<B>"	*<!--#4DHTML mavar-->*	<B>

Pour assurer une évaluation correcte de l'expression quelle que soit la langue ou la version de 4D, il est recommandé d'utiliser la syntaxe *tokenisée* pour les éléments dont le nom peut varier au fil des versions (commandes, tables, champs, constantes). Par exemple, pour insérer la commande **Heure courante**, saisissez '**Current time:C178**'. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser des tokens dans les formules**.

En cas d'erreur d'évaluation, le texte inséré sera de la forme "*<!--#4DHTML mavar-->* : ## erreur # code d'erreur".

Notes :

- L'exécution d'une méthode 4D avec *4DHTML* depuis une requête Web est soumise à la valeur de l'attribut "Disponible via les balises et URLs 4D (4DACTION...)" défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section **Sécurité des connexions**.
- Pour des raisons de sécurité, il est recommandé d'utiliser la balise 4DTEXT lorsque vous traitez des données introduites depuis l'extérieur de l'application, afin d'empêcher l'injection de code malveillant (cf. paragraphe **Notes d'utilisation** ci-dessous).

4DEVAL

Syntaxe : *<!--#4DEVAL NomVar--> ou <!--#4DEVAL Expression4D-->*
Syntaxe alternative : *$4DEVAL(NomVar)* ou *$4DEVAL(Expression4D)* (voir **Syntaxe alternative pour 4DTEXT, 4DHTML, 4DEVAL**)

La balise 4DEVAL vous permet d'évaluer une variable ou une expression 4D. Tout comme la balise 4DHTML, 4DEVAL n'échappe pas les caractères HTML lorsqu'elle retourne du texte. Cependant, à la différence de 4DHTML ou 4DTEXT, 4DEVAL vous permet d'exécuter toute instruction 4D valide, y compris des affectations ou des expressions qui ne retournent pas de valeur.

Par exemple, vous pouvez exécuter :

```
$input:="<!--#4DEVAL a:=42-->" //affectation
$input:=$input+"<!--#4DEVAL a+1-->" //calcul
TRAITER BALISES 4D($input;$output)
 //$output = "43"
```

Pour assurer une évaluation correcte de l'expression quelle que soit la langue ou la version de 4D, il est recommandé d'utiliser la syntaxe *tokenisée* pour les éléments dont le nom peut varier au fil des versions (commandes, tables, champs, constantes). Par exemple, pour insérer la commande **Heure courante**, saisissez '**Current time:C178**'. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser des tokens dans les formules**.

En cas d'erreur lors de l'interprétation, le texte inséré sera de la forme "*<!--#4DEVAL expr-->* : ## erreur # code d'erreur".

Notes :

- L'exécution d'une méthode 4D avec *4DEVAL* depuis une requête Web est soumise à la valeur de l'attribut "Disponible via les balises et URLs 4D (4DACTION...)" défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section **Sécurité des connexions**.
- Pour des raisons de sécurité, il est recommandé d'utiliser la balise 4DTEXT lorsque vous traitez des données introduites depuis l'extérieur de l'application, afin d'empêcher l'injection de code malveillant (cf. paragraphe **Notes d'utilisation** ci-dessous).

4DSCRIPT/

Syntaxe : *<!--#4DSCRIPT/NomMéthode/Param-->*

La balise *4DSCRIPT/* vous permet d'exécuter des méthodes 4D au moment du traitement du template. La présence du commentaire HTML *<!--#4DSCRIPT/NomMéthode/Param-->* provoque l'exécution de la méthode ***NomMéthode*** avec le paramètre *Param* sous forme de chaîne dans *$1*.

Note : Si la balise est appelée dans le contexte d'un process Web, au chargement de la page, 4D appelle la **Méthode base Sur authentification Web** (si elle existe). Si elle retourne **Vrai**, 4D exécute la méthode.

La méthode doit retourner du texte dans *$0*. Si la chaîne débute par le caractère de code 1, elle est considérée comme du HTML (même principe que pour la balise *4DHTML*).

Note : L'exécution d'une méthode avec *4DSCRIPT* est soumise à la valeur de l'attribut "Disponible via les balises et URLs 4D (4DACTION...)" défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section **Sécurité des connexions**.

Par exemple, vous insérez dans une page Web semi-dynamique le commentaire "**Nous sommes le <!--#4DSCRIPT/MAMETH/MONPARAM-->**". Au chargement de la page, 4D appelle la **Méthode base Sur authentification Web** (si elle existe) puis la méthode **MAMETH** en lui passant comme paramètre (dans *$1*) la chaîne "/MONPARAM".
La méthode retourne du texte dans $0 (par exemple "28/10/14"), l'expression "Nous sommes le *<!--#4DSCRIPT/MAMETH/MONPARAM-->*" devient alors "Nous sommes le 28/10/14".

Le code de MAMETH est :

```
C_TEXTE($0;$1) `Ces paramètres doivent TOUJOURS être déclarés
$0:=Chaine(Date du jour)
```

Note : Une méthode appelée par *4DSCRIPT* ne doit pas faire appel à des éléments d'interface (**DIALOGUE**, **ALERTE**...)

Comme 4D exécute les méthodes dans leur ordre d'apparition, il est tout à fait possible de faire appel à une méthode qui fixe la valeur de plusieurs variables elles-mêmes référencées plus loin dans le document, quel que soit le mode utilisé. Vous pouvez insérer autant de commentaires *<!--*

#4DSCRIPT...--> que vous voulez dans un template.

4DINCLUDE

Syntaxe : *<!--#4DINCLUDE chemin-->*

Cette balise est principalement conçue pour inclure, dans une page HTML, une autre page HTML (désignée par le paramètre *chemin*). Par défaut, seul le corps de la page HTML désignée, c'est-à-dire le contenu compris entre les balises <body> et </body>, est inclus (les balises elles-mêmes ne sont pas incluses). Ce principe permet d'éviter les conflits liés aux meta balises présentes dans les en-têtes. Toutefois, si la page HTML désignée ne contient pas de balises <body></body>, la page entière est incluse. Il vous appartient alors de veiller à la cohérence des meta balises.
Le commentaire *<!--#4DINCLUDE -->* s'avère particulièrement utile en combinaison avec les tests (*<!--#4DIF-->*) ou les boucles (*<!--#4DLOOP-->*). Il est également pratique pour inclure des bannières en fonction d'un critère, ou de manière aléatoire.
Au moment de l'inclusion, quelle que soit l'extension du nom du fichier, 4D analyse la page appelée puis insère son contenu — éventuellement modifié — dans la page d'où provient l'appel *4DINCLUDE*.

Le placement dans le cache Web d'une page incluse à l'aide du commentaire *<!--#4DINCLUDE -->* répond aux mêmes règles que celles des pages demandées via un URL ou envoyées par la commande **WEB ENVOYER FICHIER**.

Passez dans *chemin* le chemin d'accès au document à inclure. **Attention** : Dans le cas de la balise 4DINCLUDE, le chemin d'accès doit être défini relativement au document en cours d'analyse, c'est-à-dire au document "parent", et non à la racine du serveur. Utilisez la barre oblique (/) comme séparateur de dossiers et les deux-points (..) pour remonter d'un niveau hiérarchique (syntaxe HTML).

Notes :

- Lorsque vous utilisez la balise *4DINCLUDE* avec la commande **TRAITER BALISES 4D**, le dossier par défaut est le dossier contenant le fichier de structure de la base.
- Vous pouvez modifier le dossier par défaut utilisé par la balise *4DINCLUDE* dans la page courante, à l'aide de la balise *<!--#4DBASE -->* (cf. ci-dessous).

Le nombre de *<!--#4DINCLUDE chemin-->* au sein d'une page n'est pas limité. Toutefois, les appels à *<!--#4DINCLUDE chemin-->* ne peuvent s'effectuer que sur 1 niveau. Cela signifie que par exemple vous ne pouvez pas insérer le commentaire *<!--#4DINCLUDE mondoc3.html-->* dans le corps de la page mondoc2.html, elle-même appelée par *<!--#4DINCLUDE mondoc2-->* inséré dans mondoc1.html.
En outre, 4D contrôle que les inclusions ne sont pas récursives.

En cas d'erreur, le texte inséré est de la forme "*<!--#4DINCLUDE chemin-->* : Le document ne peut pas être ouvert".

Exemples :

```
<!--#4DINCLUDE souspage.html--> <!--#4DINCLUDE dossier/souspage.html--> <!--#4DINCLUDE ../dossier/souspage.html--
>
```

4DBASE

Syntaxe : *<!--#4DBASE cheminDossier-->*

La balise *<!--#4DBASE -->* permet de désigner un répertoire courant qui sera utilisé par la balise *<!--#4DINCLUDE-->*.
Lorsqu'elle est appelée dans une page Web, la balise *<!--#4DBASE -->* modifie tous les appels *<!--#4DINCLUDE-->* suivants dans cette page, jusqu'au prochain *<!--#4DBASE -->* éventuel. Si le dossier *<!--#4DBASE -->* est modifié depuis un fichier inclus, il récupère sa valeur d'origine dans le fichier parent.

Le paramètre *cheminDossier* doit contenir un chemin d'accès relatif à la page courante. Il doit se terminer par une barre oblique (/). Le dossier désigné doit être situé à l'intérieur du dossier Web.
Passez le mot-clé *WEBFOLDER* pour rétablir le chemin par défaut (relatif à la page).

Ainsi, le code suivant, devant spécifier un chemin relatif à chaque appel :

```
<!--#4DINCLUDE subpage.html--> <!--#4DINCLUDE folder/subpage1.html--> <!--#4DINCLUDE folder/subpage2.html--> <!--#4DINCLUDE
folder/subpage3.html--> <!--#4DINCLUDE  ../folder/subpage.html-->
```

... peut être réécrit à l'aide de la balise *<!--#4DBASE -->* :

```
<!--#4DINCLUDE subpage.html--> <!--#4DBASE folder/--> <!--#4DINCLUDE subpage1.html--> <!--#4DINCLUDE subpage2.html--> <!--#4DINCLUDE
subpage3.html--> <!--#4DBASE ../folder/--> <!--#4DINCLUDE subpage.html--> <!--#4DBASE WEBFOLDER-->
```

Exemple

Définition d'un répertoire pour la page d'accueil à l'aide de la balise ***<!--#4DBASE -->*** :

```
/* Index.html */ <!--#4DIF LangFR=True-->     <!--#4DBASE FR/--> <!--#4DELSE-->     <!--#4DBASE US/--> <!--
#4DENDIF--> <!--#4DINCLUDE head.html--> <!--#4DINCLUDE body.html--> <!--#4DINCLUDE footer.html-->
```

Dans le fichier head.html, le dossier courant est modifié via ***<!--#4DBASE -->***, sans que cela change sa valeur dans Index.html :

```
/* Head.htm */ /* ici le répertoire courant est relatif au fichier inclus (FR/ ou US/) */ <!--#4DBASE Styles/-->
<!--#4DINCLUDE main.css--> <!--#4DINCLUDE product.css--> <!--#4DBASE Scripts/--> <!--#4DINCLUDE main.js--> <!--
#4DINCLUDE product.js-->
```

4DCODE

La balise 4DCODE vous permet d'insérer un bloc de code 4D multi-lignes dans un *template*.

Lorsqu'une séquence "<!--#4DCODE" suivie d'un caractère espace, CR ou LF est détectée, 4D interprète directement toutes les lignes de code jusqu'à la séquence "-->" suivante. Le bloc de code lui-même peut contenir des retours chariot, des retours ligne, ou les deux ; il sera interprété séquentiellement par 4D.

Par exemple, à l'aide de la balise 4DCODE, vous pouvez écrire dans un *template* :

```
<!--#4DCODE
//Initialisation des paramètres
C_OBJECT:C1216($graphParameters)
OB SET:C1220($graphParameters;"graphType";1)
$graphType:=1
//...votre code ici
If(OB Is defined:C1231($graphParameters;"graphType")) //langage US uniquement
    $graphType:=OB GET:C1224($graphParameters;"graphType")
    If($graphType=7)
        $nbSeries:=1
        If($nbValues>8)
            DELETE FROM ARRAY:C228 ($yValuesArrPtr{1}->;9;100000)
            $nbValues:=8
        End if
    End if
End if
-->
```

Note : Dans une balise 4DCODE, le code 4D doit toujours être écrit en langage Anglais-US. 4DCODE ne tient pas compte de la préférence "Utiliser paramètres régionaux système " pour le langage 4D (voir **Langue des commandes et des constantes**).

La balise 4DCODE inclut les fonctionnalités suivantes :

- La commande **TRACE** est prise en charge et active le débogueur de 4D, vous permettant donc de déboguer le code de votre *template*.
- Toute erreur est affichée via la boîte de dialogue standard d'erreur qui propose à l'utilisateur de stopper l'exécution du code ou d'ouvrir le débogueur.
- Les lignes du texte situé entre <!--#4DCODE et --> peuvent utiliser tout type de convention de fin de ligne (cr, lf ou crlf).
- Le texte est *tokenisé* dans le contexte de la base qui a appelé la commande **TRAITER BALISES 4D**. Ce principe est important pour la reconnaissance des méthodes projet par exemple.
  Note : La propriété de méthode "Disponible via les balises HTML et les URLs 4D" n'est pas prise en compte (voir également ci-dessous **Note à propos de la sécurité**).
- Même si le texte utilise toujours la langue Anglais-US, il est recommandé d'utiliser la syntaxe avec *tokens* (:Cxxx) pour les noms des commandes et des constantes afin d'éliminer les problèmes potentiels liés au renommage des commandes ou des constantes entre deux versions de 4D.
  Note : Pour plus d'informations sur la syntaxe *:Cxxx*, veuillez vous référer à la section **Utiliser des tokens dans les formules**.

Note à propos de la sécurité : Le fait que les balises 4DCODE puissent appeler n'importe quelle commande 4D ou méthode projet pourrait être perçu comme une faille de sécurité, notamment lorsque la base est accessible via HTTP. Cependant, comme elle exécute côté serveur du code appelé depuis vos propres fichiers de templates, la balise elle-même ne pose aucun problème de sécurité. Dans ce contexte, comme pour tout serveur Web, la sécurité doit avant tout être optimale au niveau des accès distants aux fichiers du serveur.

4DIF, 4DELSE, 4DELSEIF et 4DENDIF

Syntaxe : *<!--#4DIF expression--> {<!--#4DELSEIF expression2-->...<!--#4DELSEIF expressionN-->} {<!--#4DELSE-->} <!--#4DENDIF-->*

Utilisé en conjonction avec les commentaires *<!--#4DELSEIF-->* (optionnel), *<!--#4DELSE-->* (optionnel) et *<!--#4DENDIF-->*, le commentaire *<!--#4DIF expression-->* offre la possibilité d'exécuter des portions de code de manière conditionnelle.
Le paramètre *expression* peut contenir toute expression 4D valide retournant une valeur booléenne. Elle doit figurer entre parenthèses et respecter les règles de syntaxe de 4D.

Pour assurer une évaluation correcte de l'expression quelle que soit la langue ou la version de 4D, il est recommandé d'utiliser la syntaxe *tokenisée* pour les éléments dont le nom peut varier au fil des versions (commandes, tables, champs, constantes). Par exemple, pour insérer la commande **Heure courante**, saisissez '**Current time:C178**'. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser des tokens dans les formules**.

Les blocs *<!--#4DIF expression-->* ... *<!--#4DENDIF-->* peuvent être imbriqués sur plusieurs niveaux. Comme dans 4D, chaque *<!--#4DIF expression-->* doit avoir un *<!--#4DENDIF-->* correspondant.
En cas d'erreur d'évaluation, le texte "*<!--#4DIF expression-->* : Une expression booléenne était attendue" est inséré à la place du contenu situé entre *<!--#4DIF -->* et *<!--#4DENDIF-->*.
De même, s'il n'y a pas autant de *<!--#4DENDIF-->* que de *<!--#4DIF -->*, le texte "*<!--#4DIF expression-->* : 4DENDIF attendu" est inséré à la place du contenu situé entre *<!--#4DIF -->* et *<!--#4DENDIF-->*.

A l'aide de la balise *<!--#4DELSEIF-->*, vous pouvez tester un nombre illimité de conditions. Seul le contenu suivant la première condition évaluée à **Vrai** sera exécuté. Si aucune des conditions n'est vraie, aucun contenu n'est exécuté (s'il n'y a pas de *<!--#4DELSE-->* final).
Vous pouvez utiliser une balise *<!--#4DELSE-->* après le dernier *<!--#4DELSEIF-->*. Si toutes les conditions sont fausses, les instructions suivant le *<!--#4DELSE-->* seront exécutées.

Les deux codes suivants sont équivalents.

- Code utilisant uniquement 4DELSE :

```
<!--#4DIF Condition1-->   /* Condition1 est vraie*/ <!--#4DELSE-->     <!--#4DIF Condition2-->         /* Condition2 est vraie*/
    <!--#4DELSE-->        <!--#4DIF Condition3-->            /* Condition3 est vraie */        <!--#4DELSE-->
/*Aucune condition n'est vraie*/        <!--#4DENDIF-->     <!--#4DENDIF--> <!--#4DENDIF-->
```

- Code similaire utilisant la balise 4DELSEIF :

```
<!--#4DIF Condition1-->      /* Condition1 est vraie*/ <!--#4DELSEIF Condition2-->      /* Condition2 est vraie*/ <!--#4DELSEIF
Condition3-->     /* Condition3 est vraie*/ <!--#4DELSE-->     /* Aucune condition n'est vraie */ <!--#4DENDIF-->
```

Exemple 1

Cet exemple de code inséré dans une page HTML statique affiche un libellé différent en fonction du résultat de l'expression *vnom#""* :

```
<BODY> ... <!--#4DIF (vnom#"")--> Noms commençant par <!--#4DTEXT vnom-->. <!--#4DELSE--> Aucun nom n'a été
trouvé. <!--#4DENDIF--> ... </BODY>
```

Exemple 2

Cet exemple insère des pages différentes en fonction de l'utilisateur connecté :

```
<!--#4DIF LoggedIn=False-->     <!--#4DINCLUDE Login.htm --> <!--#4DELSEIF User="Admin" -->     <!--#4DINCLUDE
AdminPanel.htm --> <!--#4DELSEIF User="Manager" -->     <!--#4DINCLUDE SalesDashboard.htm --> <!--#4DELSE-->
<!--#4DINCLUDE ItemList.htm --> <!--#4DENDIF-->
```

4DLOOP et 4ENDLOOP

Syntaxe : *<!--#4DLOOP condition--> <!--#4DENDLOOP-->*

Ce commentaire permet de répéter une portion de code tant que la condition est remplie. La portion est délimitée par *<!--#4DLOOP-->* et *<!--#4DENDLOOP-->*.
Les blocs *<!--#4DLOOP condition-->* ... *<!--#4DENDLOOP-->* peuvent être imbriqués. Comme dans 4D, chaque *<!--#4DLOOP condition-->* doit avoir un *<!--#4DENDLOOP-->* correspondant.
Il existe cinq types de conditions :

- **<!--#4DLOOP [table]-->**
  Cette syntaxe effectue une boucle pour chaque enregistrement de la sélection courante de *table* dans le process en cours. La portion de code située entre les deux commentaires est répétée pour chaque enregistrement de la sélection courante.
  Note : Lors de l'utilisation de 4DLOOP avec une table, les enregistrements sont chargés en mode Lecture seule.
  L'exemple de code suivant :

```
<!--#4DLOOP [Personnes]--> <!--#4DTEXT [Personnes]Nom--> <!--#4DTEXT [Personnes]Prenom--><BR> <!--#4DENDLOOP-->
```

  ... pourrait se traduire en langage 4D par :

```
DEBUT SELECTION([Personnes])
Tant que(Non(Fin de selection([Personnes])))
   ...
   ENREGISTREMENT SUIVANT([Personnes])
Fin tant que
```

- **<!--#4DLOOP tableau-->**
  Cette syntaxe effectue une boucle pour chaque élément du tableau. L'indice courant du tableau est incrémenté à chaque répétition de la portion de code.
  Il n'est pas possible d'utiliser cette syntaxe avec des tableaux à deux dimensions. Dans ce cas, la solution consiste à combiner une méthode et des boucles imbriquées.

  L'exemple de code suivant :

```
<!--#4DLOOP tab_noms--> <!--#4DTEXT tab_noms{tab_noms}--><BR> <!--#4DENDLOOP-->
```

  ... pourrait se traduire en langage 4D par :

```
Boucle($indice;1;Taille tableau(tab_noms))
   tab_noms:=$indice
   ...
Fin de boucle
```

- **<!--#4DLOOP méthode-->**
  Cette syntaxe effectue une boucle tant que la méthode retourne Vrai. La méthode admet un paramètre de type Entier long. Elle est appelée une première fois avec la valeur 0 pour permettre une éventuelle phase d'initialisation, puis elle est appelée successivement avec les valeurs 1, 2, 3... tant qu'elle renvoie Vrai.
  Pour des raisons de sécurité, dans le contexte d'un process Web la **Méthode base Sur authentification Web** peut être appelée, une seule fois, avant la phase d'initialisation (exécution de la méthode avec 0 comme paramètre). Si l'authentification est confirmée, la phase d'initialisation a lieu.
  En vue de la compilation de la base, il est impératif de déclarer **C_BOOLEEN**($0) et **C_ENTIER LONG**($1) au sein de la méthode.
  L'exemple de code HTML suivant :

```
<!--#4DLOOP ma_methode--> <!--#4DTEXT var--> <BR> <!--#4DENDLOOP-->
```

  ... pourrait se traduire en code 4D par :

```
Si(AuthentificationWebOK)
   Si(ma_methode(0))
      $compteur:=1
      Tant que(ma_methode($compteur))
         ...
```

```
            $compteur:=$compteur+1
         Fin tant que
      Fin de si
   Fin de si
```

La méthode *ma_methode* pourrait avoir la forme suivante :

```
C_ENTIER LONG($1)
C_BOOLEEN($0)
Si($1=0)
  `Initialisation
   $0:=Vrai
Sinon
   Si($1<50)
      ...
      var:=...
      $0:=Vrai
   Sinon
      $0:=Faux `Arrêt de la boucle
   Fin de si
Fin de si
```

- **<!--#4DLOOP expression4D-->**
  Avec cette syntaxe, la balise 4DLOOP effectue une boucle tant que l'expression 4D retourne **Vrai**. L'expression peut être toute expression booléenne valide et doit contenir une partie variable évaluée à chaque boucle afin d'éviter les boucles infinies.
  Par exemple, le code suivant :

```
<!--#4DEVAL $i:=0--> <!--#4DLOOP ($i<4)--> <!--#4DEVAL $i--> <!--#4DEVAL $i:=$i+1--> <!--#4DENDLOOP-->
```

  génère le résultat suivant :

```
0
1
2
3
```

  Pour assurer une évaluation correcte de l'expression quelle que soit la langue ou la version de 4D, il est recommandé d'utiliser la syntaxe *tokenisée* pour les éléments dont le nom peut varier au fil des versions (commandes, tables, champs, constantes). Par exemple, pour insérer la commande **Heure courante**, saisissez '**Current time:C178**'. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser des tokens dans les formules**.

- **<!--#4DLOOP pointeurTab-->**
  Dans ce cas, la balise 4DLOOP se comporte exactement comme avec un tableau : elle effectue une boucle pour chaque élément du tableau pointé. L'élément courant du tableau est incrémenté chaque fois que le morceau de code est répété.
  Cette syntaxe est principalement utile lorsque vous passez un pointeur de tableau en paramètre à la commande **TRAITER BALISES 4D**.
  Exemple :

```
TABLEAU TEXTE($array;2)
$array{1}:="hello"
$array{2}:="world"
$input:="<!--#4DEVAL $1-->"
$input:=$input+"<!--#4DLOOP $2-->"
$input:=$input+"<!--#4DEVAL $2->{$2->}--> "
$input:=$input+"<!--#4DENDLOOP-->"
TRAITER BALISES 4D($input;$output;"éléments = ";->$array)
 // $output = "éléments = hello world "
```

En cas d'erreur d'évaluation, le texte "*<!--#4DLOOP expression-->* : descriptif" est inséré à la place du contenu situé entre *<!--#4DLOOP -->* et *<!--#4DENDLOOP-->*.
Le descriptif de l'erreur peut être l'un des suivants :

- Une expression de ce type n'était pas attendue (erreur générique).
- Nom de table invalide (erreur sur le nom de la table).
- Un tableau était attendu (la variable n'est pas un tableau ou est un tableau à deux dimensions).
- La méthode n'existe pas.
- Erreur de syntaxe (lors de l'exécution de la méthode).
- Erreur de privilège (pas de droits suffisants pour accéder à la table ou à la méthode).
- 4DENDLOOP attendu (le nombre de *<!--#4DENDLOOP-->* n'est pas égal à celui de *<!--#4DLOOP -->*).

Syntaxe alternative pour 4DTEXT, 4DHTML, 4DEVAL

Plusieurs balises de transformation 4D peuvent être exprimées à l'aide d'une syntaxe utilisant le $:
$4dbalise (*expression*) peut remplacer **<!--#4dbalise *expression*-->**

Conditions de prise en charge
La syntaxe avec $ est utilisable uniquement avec certaines balises et dans certains contextes d'évaluation :

- **Balises**
  Seules les balises qui traitent et retournent des valeurs acceptent cette syntaxe :
  - 4DTEXT
  - 4DHTML
  - 4DEVAL
- **Evaluation**
  La syntaxe $ déclenche l'évaluation des balises par 4D uniquement dans les cas où l'évaluation est explicite :
  - commande **TRAITER BALISES 4D**
  - pages statiques **.shtml** servies directement par le serveur Web via des URLs
  - pages insérées via la balise 4DINCLUDE

Pour les autres balises (4DIF, 4DSCRIPT...) et dans les autres contextes d'évaluation (commandes **WEB ENVOYER FICHIER**, **WEB ENVOYER BLOB** et **WEB ENVOYER TEXTE**), la syntaxe $ n'est pas prise en charge, il est nécessaire d'utiliser la syntaxe standard.

Mode d'utilisation

Par exemple, vous pouvez écrire :

```
$4DEVAL(NomUtilisateur)
```

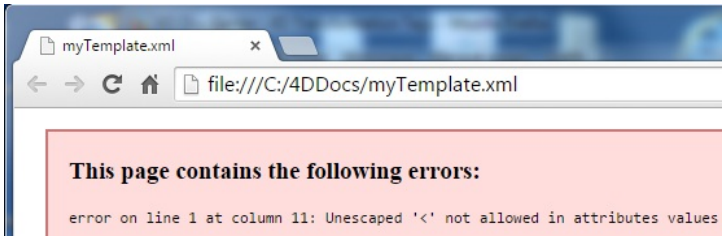
au lieu de :

```
<!--#4DEVAL(NomUtilisateur)-->
```

Le principal avantage de cette syntaxe est qu'elle vous permet d'**écrire des *templates* conformes à la norme XML**. Certains développeurs 4D ont besoin d'écrire et de valider des *templates* à base d'XML et en utilisant des éditeurs standard. Comme le caractère "<" est interdit dans une valeur d'attribut XML, il n'est pas possible d'utiliser la syntaxe "<!-- -->" pour les balises 4D sans rendre le document invalide du point du vue du XML. Parallèlement, échapper le caractère "<" empêche 4D d'interpréter correctement les balises.

Par exemple, le code suivant provoquera une erreur d'analyse XML à cause du premier caractère "<" dans la valeur de l'attribut :

```
<line x1="<!--#4DEVAL $x-->" y1="<!--#4DEVAL $graphY1-->"/>
```



Grâce à la syntaxe $, le code suivant sera en revanche parfaitement validé par l'analyseur XML :

```
<line x1="$4DEVAL($x)" y1="$4DEVAL($graphY1)"/>
```

A noter que les syntaxes *$4dbalise* et *<!--#4dbalise -->* ne sont pas strictement équivalentes : à la différence de *<!--#4dbalise -->*, le traitement de *$4dbalise* n'effectuera pas d'interprétation récursive des balises. Les balises $ sont toujours évaluées une seule fois et le résultat est considéré comme du texte brut.

Note : Pour plus d'informations sur ce point, veuillez vous référer au paragraphe **Récursivité du traitement**.

La raison de cette différence est la prévention de l'injection de code malveillant. Comme expliqué ci-dessous, il est fortement recommandé d'utiliser des balises 4DTEXT au lieu de balises 4DHTML lorsque le code manipule du texte saisi par les utilisateurs, de manière à empêcher toute réinterprétation des balises : avec 4DTEXT, les caractères spéciaux tels que "<" sont échappés, et donc toute balise 4D utilisant la syntaxe *<!--#4dbalise expression -->* perdra sa signification spécifique. Cependant, comme 4DTEXT n'échappe pas le symbole $, nous avons décidé de renoncer à prendre en charge la récursivité des traitements afin de rendre impossible toute injection de code malveillant via la syntaxe *$4dbalise (expression)*.

Les exemples suivants montrent le résultat du traitement en fonction de la syntaxe et de la balise utilisées :

```
 // exemple 1
myName:="<!--#4DHTML QUITTER 4D-->" //injection de code malveillant
input:="Mon nom est : <!--#4DHTML myName-->"
TRAITER BALISES 4D(input;output)
 //4D quitte !
```

```
 // exemple 2
myName:="<!--#4DHTML QUITTER 4D-->" //injection de code malveillant
input:="Mon nom est : <!--#4DTEXT myName-->"
TRAITER BALISES 4D(input;output)
 //output vaut "Mon nom est : &lt;!--#4DHTML QUITTER 4D--&gt;"
```

```
 // exemple 3
myName:="$4DEVAL(QUITTER 4D)" //injection de code malveillant
input:="Mon nom est : <!--#4DTEXT myName-->"
TRAITER BALISES 4D(input;output)
```

```
//output is "Mon nom est : $4DEVAL(QUIT 4D)"
```

Notez que la syntaxe *$4dbalise* prend en charge les paires fermantes de guillemets ou de parenthèses incluses. Par exemple, supposons que vous souhaitez évaluer la chaîne complexe suivante (exemple théorique) :

```
String(1) + "\"(hello)\""
```

Vous pouvez écrire :

```
input:="$4DEVAL( String(1)+\"\\\"(hello)\\\"\")"
TRAITER BALISES 4D(input;output)
-->output vaut 1"(hello)"
```

Notes d'utilisation

Récursivité du traitement

Les balises 4D sont interprétées de manière récursive : 4D tente toujours de réinterpréter le produit d'une transformation et, si une nouvelle transformation a eu lieu, une interprétation supplémentaire est effectuée, et ainsi de suite jusqu'à ce que le produit obtenu ne nécessite plus de transformation. Par exemple, soit l'instruction suivante :

```
<!--#4DHTML [Courriers]Lettre_type-->
```

Si le champ texte [Courriers]Lettre_type contient lui-même une balise, par exemple *<!--#4DSCRIPT/m_Gender-->*, elle sera évaluée récursivement après interprétation de la balise 4DHTML.

Ce principe puissant répond à la plupart des besoins liés à la transformation des textes. Attention cependant, il peut permettre également dans certains cas l'injection de code malveillant. Pour plus d'informations sur ce point, reportez-vous au paragraphe suivant.

Prévention de l'injection de code malveillant

Les balises de transformation 4D acceptent en paramètres différents types de données : textes, variables, méthodes, noms de commandes, etc. Lorsque ces données sont fournies par votre propre code, il n'y a pas de risque d'injection de code malveillant car vous contrôlez les entrées. Cependant, le code de votre base manipule souvent des données qui ont été, à un moment où à un autre, introduites via une source externe (saisie utilisateur, import, etc.).
Dans ce cas, il est conseillé de ne pas utiliser de balises de transformation qui évaluent les paramètres, comme 4DEVAL ou 4DSCRIPT, directement avec ces données.
De plus, en vertu du principe de récursivité (cf. paragraphe ci-dessus), le code malveillant peut lui-même contenir des balises de transformation. Dans ce cas, il est impératif d'utiliser la balise 4DTEXT.
Imaginez par exemple un champ de formulaire Web nommé "Nom", dans lequel l'utilisateur doit saisir son nom. Ce nom est ensuite affiché via une balise *<!--#4DHTML vNom-->* dans la page. Si un texte du type "*<!--#4DEVAL **QUITTER 4D**-->*" est inséré au lieu du nom, l'interprétation de la balise provoquera la sortie de l'application.
Pour éviter ce risque, il suffit d'utiliser systématiquement la balise 4DTEXT dans ce cas. Comme cette balise échappe les caractères HTML spéciaux, le code récursif "malin" éventuellement inséré ne sera pas réinterprété. Pour reprendre l'exemple précédent, le champ "nom" contiendra dans ce cas "*<!--#4DEVAL **QUITTER 4D**-->*" qui ne sera pas transformé.

Utilisation du "." comme séparateur décimal

A compter de la version 15 R4, 4D utilise toujours le point (.) comme séparateur décimal lors de l'évaluation d'une expression numérique via une balise 4DTEXT, 4DVAR, 4DHTML, 4DHTMLVAR ou 4DEVAL. Les paramètres régionaux sont ignorés dans ce contexte.

Ce fonctionnement facilite la maintenance et la compatibilité du code entre les diverses versions et langues de 4D.

Par exemple, quels que soient les paramètres régionaux :

```
value:=10/4
input:="<!--#4DTEXT value-->"
TRAITER BALISES 4D(input;output)
 // retourne toujours 2.5 même si les paramètres régionaux déclarent le ',' comme séparateur
```

Note de compatibilité : Si votre code converti depuis une version précédente évalue des expressions numériques via des balises 4D et en tenant compte des paramètres régionaux, il vous sera nécessaire de l'adapter à l'aide de la commande **Chaine** :

- Pour obtenir *value* avec le point comme séparateur décimal : *<!--#4DTEXT value-->*
- Pour obtenir *value* avec le séparateur décimal défini dans les paramètres régionaux : *<!--#4DTEXT Chaine(value)-->*

Caractere

Caractere (codeCaractère) -> Résultat

Paramètre	Type		Description
codeCaractère	Entier long	⇒	Code de caractère
Résultat	Chaîne	⇐	Caractère représenté par codeCaractère

Description

La fonction **Caractere** retourne le caractère dont le code est *codeCaractère*.

Passez une valeur UTF-16 (comprise entre 1 et 65535) dans *codeCaractère*.

Astuce : La fonction **Caractere** est généralement utilisée pour insérer dans l'éditeur de méthodes des caractères qui ne peuvent être saisis au clavier ou des caractères de contrôle.

Exemple

L'exemple suivant utilise la fonction **Caractere** pour insérer un retour chariot dans une boîte de dialogue d'alerte afin de séparer deux lignes d'information :

```
ALERTE("Employés : "+Chaine(Enregistrements dans table([Employés]))+Caractere(Retour chariot)+"Cliquez sur OK pour continuer.")
```

Chaine

Chaine (expression {; format {; heureComb}}) -> Résultat

Paramètre	Type		Description
expression	Expression	⇒	Expression à convertir en chaîne (peut être de type Réel, Entier, Entier long, Date, Heure, Alpha, Texte ou Booléen)
format	Chaîne, Entier long	⇒	Format d'affichage
heureComb	Heure	⇒	Heure à combiner si expression est une date
Résultat	Chaîne	⇐	expression convertie en chaîne alphanumérique

Description

La commande **Chaine** retourne sous forme de chaîne alphanumérique l'expression de type numérique, Date, Heure, chaîne ou Booléen que vous avez passée dans le paramètre *expression*.

Si vous ne passez pas le paramètre optionnel *format*, la chaîne est retournée dans le format par défaut du type de données correspondant. Si vous passez le paramètre *format*, vous pouvez définir suivant vos besoins le formatage de la chaîne retournée.

Le paramètre optionnel *heureComb* permet d'ajouter une heure à une date dans un format combiné. Il est utilisable uniquement lorsque le paramètre *expression* est une date (voir ci-dessous).

Expressions numériques

Si *expression* est du type numérique (Réel, Entier, Entier long), vous pouvez passer le paramètre optionnel de formatage de la chaîne. Voici quelques exemples :

Exemple	Résultat	Commentaire
Chaine(2^15)	"32768"	Format par défaut
Chaine(2^15;"### ##0 habitants")	"32 768 habitants"	
Chaine(1/3;"##0.00000")	"0,33333"	
Chaine(1/3)	"0,3333333333333"	Format par défaut(*)
Chaine(Arctan(1)*4)	"3,14159265359"	Format par défaut(*)
Chaine(Arctan(1)*4;"##0.00")	"3,14"	
Chaine(-1;"&x")	"0xFFFFFFFF"	
Chaine(-1;"&$")	"$FFFFFFFF"	
Chaine(0 ?+ 7;"&x")	"0x0080"	
Chaine(0 ?+ 7;"&$")	"$80"	
Chaine(0 ?+ 14;"&x")	"0x4000"	
Chaine(0 ?+ 14;"&$")	"$4000"	
Chaine(50,3;"&xml")	"50.3"	Toujours "." comme séparateur décimal
Chaine(Num(1=1);"Vrai;;Faux")	"Vrai"	
Chaine(Num(1=2);"Vrai;;Faux")	"Faux"	
Chaine(Log(-1))	""	Nombre indéfini
Chaine(1/0)	"INF"	Nombre infini positif
Chaine(-1/0)	"-INF"	Nombre infini négatif

(*) A compter de 4D v14 R3, l'algorithme de conversion des réels en texte se base sur 13 chiffres significatifs (contre 15 dans les versions précédentes de 4D).

Le format est défini de la même manière que pour un champ numérique dans un formulaire. Pour plus d'informations sur le formatage des numériques, reportez-vous à la section **Formats d'affichage** du manuel "Mode Développement" de 4D. Vous pouvez également passer le nom d'un style personnalisé dans *format*. Dans ce cas, le nom du style doit être précédé du caractère “|”.

Note : La fonction **Chaine** n'est pas compatible avec les champs de type "Entier 64 bits" en mode compilé.

Expressions de type Date

Si *expression* est de type Date, la chaîne est retournée dans le format par défaut défini dans le système. Vous pouvez passer dans le paramètre *format* une des constantes décrites ci-dessous (thème **Formats d'affichage des dates**).

Dans ce cas, vous pouvez également passer une heure dans le paramètre *heureComb*. Ce paramètre vous permet de combiner une date et une heure afin de générer des marqueurs de temps conformes aux normes en vigueur (constantes ISO Date GMT et Date RFC 1123). Ces formats sont particulièrement utiles dans le contexte des traitements XML et Web. Le paramètre *heureComb* est utilisable uniquement lorsque le paramètre *expression* est une date.

Constante	Type	Valeur	Comment
Date RFC 1123	Entier long	10	Fri, 10 Sep 2010 13:07:20 GMT
Interne date abrégé	Entier long	6	6 déc 1996
Interne date court	Entier long	7	06/12/2006
Interne date court spécial	Entier long	4	06/12/06 (mais 06/12/1896 ou 06/12/2096)
Interne date long	Entier long	5	6 décembre 2006
ISO Date	Entier long	8	2006-06-12T00:00:00 (format obsolète)
ISO Date GMT	Entier long	9	2010-09-13T16:11:53Z
Système date abrégé	Entier long	2	mer. 25 déc. 2006
Système date court	Entier long	1	06/12/2006
Système date long	Entier long	3	mercredi 6 décembre 2006
Vide si date nulle	Entier long	100	"" au lieu de 0 en cas de valeur nulle. Cette constante doit être additionnée au format d'affichage.

Note : Les formats peuvent varier en fonction des paramétrages système.

Voici quelques exemples de formats simples (en supposant que la date du jour est le jeudi 5 mars 2009) :

```
$vsRésultat:=Chaine(Date du jour) //$vsRésultat prend la valeur "05/03/09"
$vsRésultat:=Chaine(Date du jour;Interne date long) //$vsRésultat prend la valeur "5 Mars 2009"
$vsRésultat:=Chaine(Date du jour;ISO Date GMT) //$vsRésultat prend la valeur "2009-03-04T23:00:00" en France
```

Notes sur les formats combinés date/heure :

- Le format ISO Date GMT correspond à la norme ISO8601, contenant une date et une heure en tenant compte de la zone de fuseau horaire (heure GMT).

```
$madate:=Chaine(Date du jour;ISO Date GMT;Heure courante) // retourne par exemple 2010-09-13T16:11:53Z
```

  A noter le caractère "Z" final qui indique le format GMT.
  Si vous ne passez pas le paramètre *heureComb*, la commande retourne la date à minuit heure locale exprimée en heure GMT, ce qui peut entraîner un décalage :

```
$madate:=Chaine(!13/09/2010!;ISO Date GMT) // retourne 2010-09-12T22:00:00Z en France
```

- Le format ISO Date est semblable au format ISO Date GMT, à la différence près qu'il exprime la date et l'heure sans tenir compte de la zone de fuseau horaire. A noter que ce format n'étant pas conforme à la norme ISO8601, son utilisation est à réserver à des usage très spécifiques.

```
$madate:=Chaine(!13/09/2010!;ISO Date) // retourne 2010-09-13T00:00:00 quel que soit le fuseau horaire
$madate:=Chaine(Date du jour;ISO Date;Heure courante) // retourne 2010-09-13T18:11:53
```

- Le format Date RFC 1123 permet de formater un ensemble date/heure suivant la norme définie par les RFC 822 et 1123. Ce format est nécessaire par exemple pour fixer la date d'expiration des cookies dans un en-tête HTTP.

```
$madate:=Chaine(Date du jour;Date RFC 1123;Heure courante) // retourne par exemple Fri, 10 Sep 2010 13:07:20
GMT
```

  L'heure est exprimée en tenant compte de la zone de fuseau horaire (heure GMT). Si vous passez uniquement une date, la commande retourne la date à minuit heure locale exprimée en heure GMT, ce qui peut entraîner un décalage :

```
$madate:=Chaine(Date du jour;Date RFC 1123) // retourne Thu, 09 Sep 2010 22:00:00 GMT
```

Expressions de type Heure
Si *expression* est de type Heure, la chaîne est retournée dans le format par défaut *hh:mm:ss*. Vous pouvez passer dans le paramètre *format* une des constantes suivantes (thème **Formats d'affichage des heures**) :

Constante	Type	Valeur	Comment
h mn	Entier long	2	01:02
h mn Matin Après Midi	Entier long	5	1:02 du matin
h mn s	Entier long	1	01:02:03
Heures minutes	Entier long	4	1 heure 2 minutes
Heures minutes secondes	Entier long	3	1 heure 2 minutes 3 secondes
ISO heure	Entier long	8	0000-00-00T01:02:03
Minutes secondes	Entier long	7	62 minutes 3 secondes
mn s	Entier long	6	62:03
Système heure court	Entier long	9	01:02:03
Système heure long	Entier long	11	1:02:03 AM HNEC (Mac uniquement)
Système heure long abrégé	Entier long	10	1•02•03 AM (Mac uniquement)
Vide si heure nulle	Entier long	100	"" au lieu de 0

Notes :

- Le format ISO heure correspond à la norme ISO8601, contenant en principe une date et une heure. Ce format ne prenant pas en charge les dates/heures combinées, la partie date est remplie avec des 0. Ce format exprime l'heure locale.
- La constanteVide si heure nulle doit être additionnée au format, elle indique qu'en cas de valeur nulle 4D doit retourner une chaîne vide au lieu de zéros.

Voici quelques exemples (en supposant qu'il soit 17h30 et 45 secondes) :

```
$vsRésultat:=Chaine(Heure courante) ` $vsRésultat prend la valeur "17:30:45"
$vsRésultat:=Chaine(Heure courante;Heures minutes secondes)
 ` $vsRésultat prend la valeur "17 heures 30 minutes 45 secondes"
```

Expressions de type chaîne
Si *expression* est de type Alpha ou Texte, la commande retourne la même valeur que celle passée en paramètre. Ce fonctionnement est utile notamment dans le cadre d'une programmation générique utilisant des pointeurs.
Dans ce cas, le paramètre *format*, s'il est passé, est ignoré.

Expressions de type Booléen
Si *expression* est de type Booléen, la commande retourne la chaîne "Vrai" ou "Faux" dans la langue de l'application ("True" ou "False" dans une version anglaise de 4D).
Dans ce cas, le paramètre *format*, s'il est passé, est ignoré.

Code de caractere

Code de caractere (unCaractère) -> Résultat

Paramètre	Type		Description
unCaractère	Chaîne	➡	Caractère dont vous voulez obtenir le code
Résultat	Entier long	⮌	Code du caractère

Description

La commande **Code de caractere** retourne le code Unicode UTF-16 (compris entre 1 et 65535) de *unCaractère*.

Si la chaîne *unCaractère* comporte plus d'un caractère, **Code de caractere** retourne uniquement le code du premier caractère.

La fonction **Code de caractere** est l'inverse de **Caractere**. Elle retourne le caractère désigné par un code UTF-16.

Exemple 1

Les caractères majuscules et minuscules ne sont pas différenciés lors d'une comparaison ou d'une recherche. Vous pouvez utiliser la fonction **Code de caractere** si vous souhaitez établir une distinction entre les caractères majuscules et les minuscules.
En effet, cette ligne retourne **VRAI** :

```
("A"="a")
```

En revanche, cette ligne retourne **FAUX** :

```
(Code de caractere("A")=Code de caractere("a"))
```

Exemple 2

L'exemple suivant retourne le code du premier caractère de la chaîne *"ABC"* :

```
RécupCode:=Code de caractere("ABC") ` RécupCode prend la valeur 65, le code de caractère de A
```

Exemple 3

Le code suivant :

```
Boucle($vlCar;1;Longueur(vtText))
   Au cas ou
      :(vtText[[$vlCar]]=Caractere(Retour chariot))
   ` Faire quelque chose
      :(vtText[[$vlCar]]=Caractere(Tab))
   ` Faire autre chose
      :(...)
   ` ...
   Fin de cas
Fin de boucle
```

... lorsqu'il est utilisé de nombreuses fois avec des textes de taille importante, s'exécutera plus vite, une fois compilé, s'il est écrit ainsi :

```
Boucle($vlCar;1;Longueur(vtText))
   $vlCode:=Code de caractere(vtText[[$vlCar]])
   Au cas ou
      :($vlCode=Retour chariot)
   ` Faire quelque chose
      :($vlCode=Tabulation)
   ` Faire autre chose
      :(...)
   ` ...
   Fin de cas
Fin de boucle
```

... et ce, pour deux raisons principales : il ne référence un caractère qu'une seule fois par itération, et compare des entiers longs et non des chaînes de caractères lorsqu'il teste la présence de retours chariot et de tabulations. Nous vous conseillons d'employer cette technique lorsque vous travaillez avec des caractères standard tels que des Retours chariot et des Tabulations.

CONVERTIR DEPUIS TEXTE

CONVERTIR DEPUIS TEXTE (texte4D ; jeuCaractères ; blobConverti)

Paramètre	Type		Description
texte4D	Chaîne	⇒	Texte exprimé dans le jeu de caractères courant de 4D
jeuCaractères	Chaîne, Entier long	⇒	Nom ou Numéro de jeu de caractères
blobConverti	BLOB	⇐	BLOB contenant le texte converti

Description

La commande **CONVERTIR DEPUIS TEXTE** permet de convertir un texte exprimé dans le jeu de caractères courant de 4D en un texte exprimé dans un autre jeu de caractères.

Passez dans le paramètre *texte4D* le texte devant être converti. Ce texte est exprimé dans le jeu de caractères de 4D. Dans les bases de données créées à partir de la version 11, 4D utilise le jeu de caractères Unicode par défaut.

Passez dans *jeuCaractères* le jeu de caractères à utiliser pour la conversion. Vous pouvez passer une chaîne contenant le nom standard du jeu (par exemple “ISO-8859-1” ou “UTF-8”) ou son identifiant MIBEnum. Voici la liste des jeux de caractères pris en charge par les commandes **CONVERTIR DEPUIS TEXTE** et **Convertir vers texte** :

MIBEnum	Nom(s)
1017	UTF-32
1018	UTF-32BE
1019	UTF-32LE
1015	UTF-16
1013	UTF-16BE
1014	UTF-16LE
106	UTF-8
1012	UTF-7
3	US-ASCII
3	ANSI_X3.4-1968
3	ANSI_X3.4-1986
3	ASCII
3	cp367
3	csASCII
3	IBM367
3	iso-ir-6
3	ISO_646.irv:1991
3	ISO646-US
3	us
2011	IBM437
2011	cp437
2011	437
2011	csPC8CodePage437
2028	ebcdic-cp-us
2028	cp037
2028	csIBM037
2028	ebcdic-cp-ca
2028	ebcdic-cp-n
2028	ebcdic-cp-wt
2028	IBM037
2027	MacRoman
2027	x-mac-roman
2027	mac
2027	macintosh
2027	csMacintosh
2252	windows-1252
1250	MacCE
1250	x-mac-ce
2250	windows-1250
1251	x-mac-cyrillic
2251	windows-1251
1253	x-mac-greek
2253	windows-1253
1254	x-mac-turkish
2254	windows-1254
1256	x-mac-arabic
2256	windows-1256
1255	x-mac-hebrew
2255	windows-1255

1257	x-mac-ce
2257	windows-1257
17	Shift_JIS
17	csShiftJIS
17	MS_Kanji
17	Shift-JIS
39	ISO-2022-JP
39	csISO2022JP
2026	Big5
2026	csBig5
38	EUC-KR
38	csEUCKR
2084	KOI8-R
2084	csKOI8R
4	ISO-8859-1
4	CP819
4	csISOLatin1
4	IBM819
4	iso-ir-100
4	ISO_8859-1
4	ISO_8859-1:1987
4	l1
4	latin1
5	ISO-8859-2
5	csISOLatin2
5	iso-ir-101
5	ISO_8859-2
5	ISO_8859-2:1987
5	l2
5	latin2
6	ISO-8859-3
6	csISOLatin3
6	ISO-8859-3:1988
6	iso-ir-109
6	ISO_8859-3
6	l3
6	latin3
7	ISO-8859-4
7	csISOLatin4
7	ISO-8859-4:1988
7	iso-ir-110
7	ISO_8859-4
7	l4
7	latin4
8	ISO-8859-5
8	csISOLatinCyrillic
8	cyrillic
8	ISO-8859-5:1988
8	iso-ir-144
8	ISO_8859-5
9	ISO-8859-6
9	arabic
9	ASMO-708
9	csISOLatinArabic
9	ECMA-114
9	ISO-8859-6:1987
9	iso-ir-127
9	ISO_8859-6
10	ISO-8859-7
10	csISOLatinGreek
10	ECMA-118
10	ELOT_928
10	greek
10	greek8
10	iso-ir-126
10	ISO_8859-7
10	ISO_8859-7:1987
11	ISO-8859-8
11	csISOLatinHebrew
11	hebrew

11	iso-ir-138
11	ISO_8859-8
11	ISO_8859-8:1988
12	ISO-8859-9
12	csISOLatin5
12	iso-ir-148
12	ISO_8859-9
12	ISO_8859-9:1989
12	l5
12	latin5
13	ISO-8859-10
13	csISOLatin6
13	iso-ir-157
13	ISO_8859-10
13	ISO_8859-10:1992
13	l6
13	latin6
109	ISO-8859-13
111	ISO-8859-15
111	Latin-9
113	GBK
2025	GB2312
2025	csGB2312
2025	x-mac-chinesesimp
2024	Windows-31J
57	GB_2312-80
57	csISO58GB231280

Note : Plusieurs lignes ont le même identifiant MIBEnum car un jeu de caractères peut avoir plusieurs noms (alias).

Pour plus d'informations sur les noms des jeux de caractères, reportez-vous à l'adresse http://www.iana.org/assignments/character-sets

Après l'exécution de la commande, le texte converti est retourné dans le BLOB *blobConverti*. Ce BLOB pourra être relu par la commande **Convertir vers texte**.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable OK prend la valeur 1. Sinon, elle prend la valeur 0.

Convertir vers texte

Convertir vers texte (blob ; jeuCaractères) -> Résultat

Paramètre	Type		Description
blob	BLOB	➡	BLOB contenant un texte exprimé dans un jeu de caractères spécifique
jeuCaractères	Chaîne, Entier long	➡	Nom ou Numéro du jeu de caractères de blob
Résultat	Texte	⮌	Contenu de blob exprimé dans le jeu de caractères 4D

Description

La commande **Convertir vers texte** convertit le texte contenu dans le paramètre *blob* et le retourne en texte exprimé dans le jeu de caractères de 4D. 4D utilise le jeu de caractères UTF-16 par défaut.

Passez dans *jeuCaractères* le jeu de caractères dans lequel est exprimé le texte contenu dans *blob*, et qui doit être utilisé pour la conversion. Si le conteneur de données contient du texte copié depuis 4D, le jeu de caractères du BLOB sera probablement UTF-16. Vous pouvez passer une chaîne fournissant le nom standard du jeu ou l'un de ses alias (par exemple “ISO-8859-1” ou “UTF-8”), ou encore son identifiant (entier long). Pour plus d’informations, reportez-vous à la description de la commande **CONVERTIR DEPUIS TEXTE**.

Convertir vers texte prend en charge les BOM (Byte Order Mark). Si le jeu de caractères spécifié est de type Unicode (UTF-8, UTF-16 ou UTF-32), 4D tente d’identifier une BOM parmi les premiers octets reçus. Si elle est détectée, elle est filtrée du résultat et 4D utilise le jeu de caractères qu’elle définit au lieu du jeu de caractères spécifié.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable OK prend la valeur 1. Sinon, elle prend la valeur 0.

Inserer chaine

Inserer chaine (source ; insertion ; positionDépart) -> Résultat

Paramètre	Type		Description
source	Chaîne	⇒	Chaîne dans laquelle effectuer l'insertion
insertion	Chaîne	⇒	Chaîne à insérer dans source
positionDépart	Entier long	⇒	Position de l'insertion
Résultat	Chaîne	⇐	Chaîne résultante

Description

Inserer chaine insère la chaîne de caractères alphanumériques *insertion* dans la chaîne *source* à partir de *position* et retourne la chaîne de caractères résultante. La chaîne *insertion* est placée avant le caractère désigné par *position*.

Si *insertion* est une chaîne vide (""), **Inserer chaine** retourne *source* inchangé.

Si *position* est supérieur à la longueur de *source*, *insertion* est ajouté à la fin de *source*. Si *position* est inférieur à un (1), *insertion* est inséré au début de *source*.

Inserer chaine est différent de **Remplacer caracteres** puisque cette fonction insère des caractères au lieu de les remplacer.

Exemple

L'exemple suivant illustre l'utilisation de **Inserer chaine**. Les résultats sont affectés à la variable *vRésultat*.

```
vRésultat:=Inserer chaine("L'arbre";" vert";8) ` vRésultat est égal à "L'arbre vert"
vRésultat:=Inserer chaine("Tale";"b";3) ` vRésultat est égal à "Table"
vRésultat:=Inserer chaine("Indention";"ta";6) ` vRésultat est égal à "Indentation"
```

LIRE MOTS CLES TEXTE

LIRE MOTS CLES TEXTE (texte ; tabMotsClés {; *})

Paramètre	Type		Description
texte	Texte	➡	Texte original
tabMotsClés	Tableau texte	⬅	Tableau contenant les mots-clés
*	Opérateur	➡	Si passé = mots uniques

Description

La commande **LIRE MOTS CLES TEXTE** découpe la totalité du *texte* en mots et crée, pour chaque mot obtenu, un élément dans le tableau texte *tabMotsClés*.

Le découpage en mots est effectué à l'aide du même algorithme que celui que 4D utilise pour construire les **Index de mots-clés**. Cet algorithme est basé sur la librairie ICU. Pour plus d'informations sur les séparateurs pris en compte, reportez-vous à l'adresse suivante : http://userguide.icu-project.org/boundaryanalysis.

Note : A la demande des utilisateurs, une exception a été introduite pour les langages français et italien : le caractère apostrophe ' suivi d'une voyelle ou de la lettre h est considéré comme séparateur de mot. Par exemple, les chaînes "L'homme" ou "l'arbre" seront bien découpées en "L'"+"homme" et "l'"+"arbre".

L'algorithme utilisé diffère si l'option **N'utiliser que les caractères non alphanumériques pour les mots-clés** est cochée ou non dans les Propriétés de la base (reportez-vous à la section **Page Base de données/Stockage des données** dans le manuel *Mode Développement*).

Passez dans le paramètre *texte* le texte original à découper. Ce texte peut être stylé, dans ce cas les balises de style sont simplement ignorées.

Passez dans le paramètre *tabMotsClés* le tableau texte qui sera rempli par la commande avec les mots extraits du texte.

Si vous passez le paramètre optionnel *, la commande ne stockera chaque mot-clé qu'une seule fois dans *tabMotsClés*. Par défaut, si ce paramètre est omis, tous les mots extraits du texte sont stockés dans le tableau, même s'ils apparaissent plusieurs fois.

Cette commande permet d'effectuer de façon simple des recherches parmi des enregistrements contenant des textes de grande taille, en ayant la garantie d'utiliser les mêmes mots-clés que 4D. Par exemple, soit un texte contenant "10.000 Jean-Pierre BC45". Si le découpage en mots-clés donne "10.000" + "Jean" + "Pierre" + "BC45", le tableau contiendra 4 éléments. Par programmation, il est alors facile d'effectuer une boucle dans ce tableau afin de trouver les enregistrements contenant un ou plusieurs de ces mots-clés à l'aide de l'opérateur % (voir exemples).

Exemple 1

Dans un formulaire contenant une zone de recherche, l'utilisateur peut saisir un ou plusieurs mot(s). Lorsqu'il valide, on recherche les enregistrements dont le champ *MonChamp* contient au moins un des mots saisis par l'utilisateur.

```
 // vSearch est la variable de la zone de saisie dans le formulaire
LIRE MOTS CLES TEXTE(vSearch;tSearch;*)
 //* pour le cas où l'utilisateur saisirait le même mot plusieurs fois
NOMMER ENSEMBLE([MaTable];"Globaltrouve")
$n:=Taille tableau(tSearch)
Boucle($i;1;$n)
   CHERCHER([MaTable];[MaTable]MonChamp% tSearch{$i})
   NOMMER ENSEMBLE([MaTable];"trouve")
   REUNION("Globaltrouve";"trouve";"Globaltrouve")
Fin de boucle
UTILISER ENSEMBLE("Globaltrouve")
```

Exemple 2

Dans le même formulaire que précédemment, on recherche les enregistrements dont le champ *MonChamp* contient tous les mots saisis par l'utilisateur.

```
 // vSearch est la variable de la zone de saisie dans le formulaire
LIRE MOTS CLES TEXTE(vSearch;tSearch;*)
$n:=Taille tableau(tSearch)
CHERCHER([MaTable];[MaTable]ID >=0;*)
 // initialiser la recherche = tous les enregistrements
Boucle($i;1;$n)
   CHERCHER([MaTable];&;[MaTable]MonChamp% tSearch{$i};*)
 // ajouter le critère
Fin de boucle
CHERCHER([MaTable]) //recherche
```

Exemple 3

Pour compter les mots d'un texte :

```
LIRE MOTS CLES TEXTE(vTexte;tMots) // tous les mots
$n:=Taille tableau(tMots)
LIRE MOTS CLES TEXTE(vTexte;tMots;*) // mots différents
$m:=Taille tableau(tMots)
ALERTE("Ce texte contient "+Chaine($n)+" mots distincts parmi "+Chaine($m))
```

Lire traduction chaine

Lire traduction chaine (resName) -> Résultat

Paramètre	Type		Description
resName	Chaîne	⇨	Nom d'attribut resname
Résultat	Chaîne	⇦	Valeur de la chaîne désignée par resName dans le langage courant

Description

La commande **Lire traduction chaine** retourne la valeur de la chaîne désignée par l'attribut *resName* pour la langue courante.

Cette commande fonctionne uniquement dans le cadre d'une architecture XLIFF. Pour plus d'informations sur ce type d'architecture, reportez-vous à la description de la prise en charge du XLIFF dans le manuel *Mode Développement*.

Note : La commande **Lire langue base** permet de connaître la langue utilisée par l'application.

Passez dans *resName* le nom de ressource de la chaîne dont vous voulez obtenir la traduction dans la langue cible courante (target).

A noter que le XLIFF est diacritique.

Exemple

Voici un extrait de fichier .xlf :

```
<file source-language="en-US" target-language="fr-FR">    [...]    <trans-unit resname="Show on disk">
      <source>Show on disk</source>       <target>Montrer sur le disque</target>    </trans-unit>
```

Après exécution de l'instruction suivante :

```
$valeurFR:=Lire traduction chaine("Show on disk")
```

... si la langue courante est le français, $valeurFR contient "Montrer sur le disque".

Variables et ensembles système

Si la commande a été exécutée correctement, la variable OK prend la valeur 1. Si *resName* n'est pas trouvé, la commande retourne une chaîne vide et la variable OK prend la valeur 0.

Longueur

Longueur (chaîne) -> Résultat

Paramètre	Type		Description
chaîne	Chaîne	⇒	Chaîne dont vous voulez connaître la longueur
Résultat	Entier long	⇐	Nombre de caractères de chaîne

Description

Longueur vous permet d'obtenir la longueur de *laChaîne*. **Longueur** retourne le nombre de caractères alphanumériques contenus dans *laChaîne*.

Note : En mode Unicode, si vous souhaitez vérifier qu'une chaîne ne contient aucun caractère, y compris des caractères ignorables, vous devez utiliser le test Si(Longueur(vTexte)=0) plutôt que Si(vTexte=""). En effet, si la chaîne contient par exemple Caractere(1) qui est un caractère ignorable, Longueur(vTexte) retourne bien 1 mais vTexte="" retourne Vrai.

Exemple

L'exemple suivant illustre l'utilisation de **Longueur**. Les valeurs retournées sont assignées à la variable *vRésultat*. Les commentaires fournissent la valeur de *vRésultat* :

```
vRésultat:=Longueur("Topaze") ` vRésultat prend la valeur 6
vRésultat:=Longueur("Citoyen") ` vRésultat prend la valeur 7
```

Majusc

Majusc (laChaîne {; *}) -> Résultat

Paramètre	Type		Description
laChaîne	Chaîne	→	Chaîne à convertir en majuscules
*	Opérateur	→	Si passé : conserver les accents Si omis : supprimer les accents
Résultat	Chaîne	←	chaîne en majuscules

Description

Majusc retourne une chaîne de caractères égale à *laChaîne* dont tous les caractères alphabétiques ont été convertis en majuscules.

Le paramètre facultatif *, s'il est passé, indiqueque les éventuels caractères accentués présents dans *laChaîne* doivent être retournés sous forme de majuscules accentuées. Par défaut, lorsque ce paramètre est omis, les caractères accentués “perdent” leur accent une fois la conversion effectuée.

Exemple 1

Cet exemple compare les résultats obtenus suivant que le paramètre * a été passé ou non :

```
$lachaine:=Majusc("hélène") ` $lachaine vaut « HELENE »
$lachaine:=Majusc("hélène";*) ` $lachaine vaut « HÉLÈNE »
```

Exemple 2

Reportez-vous à l'exemple de **Minusc**.

Minusc

Minusc (laChaîne {; *}) -> Résultat

Paramètre	Type		Description
laChaîne	Chaîne	→	Chaîne à convertir en minuscules
*	Opérateur	→	Si passé : conserver les accents Si omis : supprimer les accents
Résultat	Chaîne	←	chaîne en minuscules

Description

Minusc retourne une chaîne de caractères égale à *laChaîne* dont tous les caractères alphabétiques ont été convertis en minuscules.

Le paramètre facultatif *, s'il est passé, indique que les éventuels caractères accentués présents dans *laChaîne* doivent être retournés sous forme de minuscules accentuées. Par défaut, lorsque ce paramètre est omis, les caractères accentués “perdent” leur accent une fois la conversion effectuée.

Exemple 1

L'exemple suivant est une méthode projet qui met en majuscule (capitale) le premier caractère de la chaîne ou du texte qui lui est passé(e). Par exemple, Nom := Capitale ("jean") donnerait à *Nom* la valeur "Jean" :

```
   ` Méthode projet Capitale
   ` Capitale ( Chaîne ) -> Chaîne
   ` Capitale ( Tout texte ou chaîne ) -> texte avec une lettre capitale

$0:=Minusc($1)
Si(Longueur($0)>0)
   $0≤1≥:=Majusc($0≤1≥)
Fin de si
```

Exemple 2

Cet exemple compare les résultats obtenus suivant que le paramètre * a été passé ou non :

```
$lachaine:=Minusc("DÉJÀ VU") ` $lachaine vaut « deja vu »
$lachaine:=Minusc("DÉJÀ VU";*) ` $lachaine vaut « déjà vu »
```

Num

Num (expression {; séparateur}) -> Résultat

Paramètre	Type		Description
expression	Chaîne, Booléen, Entier long	⇒	Chaîne à convertir en numérique ou Booléen à convertir en 0 ou 1 ou Expression numérique
séparateur	Chaîne	⇒	Séparateur décimal
Résultat	Entier long	⇐	Valeur numérique du paramètre expression

Description

La fonction **Num** retourne sous forme de numérique l'expression de type chaîne, Booléen ou numérique que vous avez passée dans le paramètre *expression*. Le paramètre facultatif *séparateur* permet de désigner un séparateur décimal pour l'évaluation des expressions de type chaîne.

Expressions de type chaîne

Si *expression* ne contient que des caractères alphabétiques, **Num** retourne zéro. Si *expression* contient des caractères alphabétiques et des caractères numériques, **Num** ignore les caractères alphabétiques. Ainsi, **Num** transformera la chaîne "a1b2c3" en nombre 123.

Il existe trois caractères réservés que **Num** traite de manière particulière. Il s'agit du séparateur décimal tel que défini dans le système (si le paramètre *séparateur* n'est pas passé), du tiret (-) et du *e* (ou **E**). Ils seront interprétés en tant que caractères de formatage des nombres :

- Le séparateur décimal est interprété en tant que tel et doit être inclus dans la chaîne de caractères numériques. Par défaut, la commande utilise le séparateur décimal défini dans le système d'exploitation. Vous pouvez modifier ce caractère à l'aide du paramètre *séparateur* (cf. ci-dessous).
- Le tiret définit un nombre ou un exposant négatif (signe moins). Le tiret doit être placé devant tout caractère numérique négatif ou derrière le e pour un exposant. Hormis le cas du caractère e, si le tiret est inclus dans une chaîne numérique, la partie de la chaîne se trouvant derrière le tiret est ignorée. Par exemple, **Num("123-456")** retourne 123, mais **Num("-9")** retourne -9.
- Le e ou E désigne tout caractère numérique se trouvant à sa droite comme étant la puissance d'un exposant. Le e doit être inclus dans une chaîne numérique. Ainsi, **Num ("123e-2")** retourne 1,23.
  A noter que dans le cas où la chaîne comporte plus d'un caractère e, la conversion pourra donner des résultats différents sous Mac OS et sous Windows.

Le paramètre *séparateur* permet de désigner un séparateur décimal personnalisé pour l'évaluation de *expression*. Lorsque la chaîne à évaluer est exprimée avec un séparateur décimal différent du séparateur système, la commande retourne un résultat incorrect. Le paramètre *séparateur* permet dans ce cas d'obtenir une évaluation correcte. Lorsque ce paramètre est passé, la commande ne tient pas compte du séparateur décimal système. Vous pouvez passer un ou plusieurs caractères.

Note : La commande **LIRE FORMATAGE SYSTEME** permet de connaître le séparateur décimal courant ainsi que plusieurs autres paramètres système régionaux.

Expressions de type Booléen

Si vous passez une expression booléenne dans le paramètre *expression*, **Num** retourne 1 si *expression* est VRAI, sinon **Num** retourne 0.

Expressions de type numérique

Si vous passez une expression numérique dans le paramètre *expression*, **Num** retourne telle quelle la valeur passée dans le paramètre *expression*. Ce fonctionnement est utile notamment dans le cadre d'une programmation générique utilisant des pointeurs.

Exemple 1

L'exemple suivant illustre la manière dont **Num** fonctionne lorsqu'un argument de type chaîne lui est passé. A chaque ligne, un numérique est assigné à la variable *vRésultat*. Les commentaires décrivent les résultats :

```
vRésultat:=Num("ABCD") ` vRésultat vaut 0
vRésultat:=Num("A1B2C3") ` vRésultat vaut 123
vRésultat:=Num("123") ` vRésultat vaut 123
vRésultat:=Num("123,4") ` vRésultat vaut 123,4
vRésultat:=Num("-123") ` vRésultat vaut -123
vRésultat:=Num("-123e2") ` vRésultat vaut -12300
```

Exemple 2

Dans l'exemple suivant, *[Client]Dette* est comparé à la valeur 1000. La fonction **Num** appliquée à cette comparaison retourne 0 ou 1. La multiplication d'une chaîne par 0 ou 1 retourne soit la chaîne, soit une chaîne vide. En définitive, le champ *[Client]Risque* reçoit la valeur "Acceptable" ou "Inacceptable" :

```
  ` Si le client a des dettes inférieures à 1000, le risque est acceptable.
  ` Si le client a des dettes supérieures à 1000, le risque est inacceptable.
[Client]Risque:=("Acceptable"*Num([Client]Dettes<1000))+("Inacceptable"*Num([Client]Dettes>=1000))
```

Exemple 3

Cet exemple compare les résultats obtenus en fonction du séparateur "courant" :

```
$lachaine:="33,333.33"
$lenum:=Num($lachaine)
  ` par défaut, $lenum vaut 33,33333 sur un système français
$lenum:=Num($lachaine;".")
  ` $lenum vaut bien 33 333,33 quel que soit le système
```

Position

Position (àChercher ; laChaîne {; début {; longTrouvée}}{; *}) -> Résultat

Paramètre	Type		Description
àChercher	Chaîne	⇒	Chaîne à rechercher
laChaîne	Chaîne	⇒	Chaîne dans laquelle effectuer la recherche
début	Entier long	⇒	Position dans laChaîne où débuter la recherche
longTrouvée	Entier long	⇐	Longueur de la chaîne trouvée
*	Opérateur	⇒	Si passé : évaluation basée sur les codes de caractères
Résultat	Entier long	⇐	Position de la première occurrence de àChercher

Description

Position retourne la position de la première occurrence de *àChercher* dans *laChaîne*.

Si *laChaîne* ne contient pas *àChercher*, la fonction retourne zéro (0).

Si **Position** trouve une occurrence de *àChercher*, la fonction retourne la position du premier caractère de cette occurrence dans *laChaîne*.
Si vous demandez la position d'une chaîne vide à l'intérieur d'une chaîne vide, **Position** retourne zéro (0).

Par défaut, la recherche débute au premier caractère de *laChaîne*. Le paramètre facultatif *début* vous permet de préciser le caractère auquel doit démarrer la recherche dans *laChaîne*.

Le paramètre *longTrouvée*, s'il est passé, retourne la longueur de la chaîne effectivement trouvée par la recherche. Ce paramètre est nécessaire pour pouvoir gérer correctement les lettres pouvant s'écrire à l'aide d'un ou plusieurs caractères (ex : æ et ae, ß et ss...).
A noter que lorsque le paramètre * est passé (cf. ci-dessous), ces lettres ne sont pas considérées comme équivalentes (æ # ae) ; dans ce mode, *longTrouvée* est toujours égal à la longueur de *àChercher* (si une occurrence est trouvée).

Par défaut, la commande effectue des comparaisons globales, tenant compte des particularités linguistiques et des lettres pouvant s'écrire avec un ou plusieurs caractères (par exemple æ = ae). En revanche, elle n'est pas diacritique (a=A, a=à...) et ne tient pas compte des caractères "ignorables". Les caractères ignorables comprennent tous les caractères du subset unicode *C0 Control* (U+0000 à U+001F, ascii character control set) à l'exception des caractères imprimables (U+0009 TAB, U+0010 LF, U+0011 VT, U+0012 FF and U+0013 CR).
Pour modifier ce fonctionnement, passez l'astérisque * en dernier paramètre. Dans ce cas, les comparaisons sont effectuées sur la base des codes des caractères. Vous devez donc passer le paramètre * :

- si vous souhaitez tenir compte des caractères spéciaux, utilisés par exemple comme délimiteurs (**Caractere**(1)...),
- si l'évaluation des caractères doit tenir compte de la casse et des accents (a#A, a#à...).
  A noter que dans ce mode, l'évaluation ne gère pas les variations d'écriture des mots.

Note : Dans certains cas, l'utilisation du paramètre * peut accélérer sensiblement l'exécution de la commande.

Attention : Vous ne pouvez pas utiliser le caractère joker (@) avec **Position**. Si, par exemple, vous passez *"abc@"* dans *àChercher*, la fonction recherchera effectivement la chaîne *"abc@"* et non pas "abc suivi de toute valeur".

Exemple 1

Les exemples suivants illustrent l'utilisation de **Position**. Les résultats sont assignés à la variable *vRésultat*. Les commentaires fournissent la valeur de *vRésultat* :

```
vRésultat:=Position("ll";"Billard") ` vRésultat prend la valeur 3
vRésultat:=Position(vText1;vText2) ` Position de la première occurrence de vText1 dans vText2
vRésultat:=Position("day";"Today is the first day";1) ` vRésultat prend la valeur 3
vRésultat:=Position("day";"Today is the first day";4) ` vRésultat prend la valeur 20
vRésultat:=Position("DAY";"Today is the first day";1;*) ` vRésultat prend la valeur 0
vRésultat:=Position("oe";"Nœud";1;$long) ` vRésultat =2, $long = 1
```

Exemple 2

Dans l'exemple suivant, le paramètre *longTrouvée* permet de rechercher toutes les occurrences de "fluss" dans un texte, quelle que soit l'orthographe du mot :

```
$départ:=1
Repeter
   vRésultat:=Position("fluss";$letexte;$départ;$longtrouvée)
   $départ:=$départ+$longtrouvée
Jusque(vRésultat=0)
```

Remplacer caracteres

Remplacer caracteres (source ; nouveau ; positionDépart) -> Résultat

Paramètre	Type		Description
source	Chaîne	→	Chaîne de départ
nouveau	Chaîne	→	Nouveaux caractères
positionDépart	Entier long	→	Position de départ du remplacement
Résultat	Chaîne	↩	Chaîne résultante

Description

Remplacer caracteres retourne une chaîne résultant du remplacement des caractères, dans la chaîne *source*, à partir de *positionDépart*, par la chaîne *nouveau*.

Si *nouveau* est une chaîne vide (""), **Remplacer caracteres** retourne *source* inchangé. **Remplacer caracteres** retourne toujours une chaîne de la même longueur que *source*. Si *positionDépart* est inférieur ou supérieur à la longueur de *source*, **Remplacer caracteres** retourne *source*.

La fonction **Remplacer caracteres** est différente de **Inserer chaine** car elle remplace des caractères au lieu de les insérer.

Exemple

L'exemple suivant illustre l'utilisation de **Remplacer caracteres**. Les résultats sont affectés à la variable *vRésultat*.

```
vRésultat:=Remplacer caracteres("Acme";"CME";2) ` vRésultat est égal à "ACME"
vRésultat:=Remplacer caracteres("novembre";"déc";1) ` vRésultat est égal à "décembre"
```

Remplacer chaine

Remplacer chaine (source ; obsolète ; nouveau {; combien}{; *}) -> Résultat

Paramètre	Type		Description
source	Chaîne	⇒	Chaîne de départ
obsolète	Chaîne	⇒	Caractère(s) à remplacer
nouveau	Chaîne	⇒	Chaîne de remplacement (si chaîne vide, toutes les occurrences sont effacées)
combien	Entier long	⇒	Nombre de remplacements à effectuer
*	Opérateur	⇒	Si passé : évaluation basée sur les codes de caractères
Résultat	Chaîne	⇐	Chaîne résultante

Description

Remplacer chaine retourne une chaîne de caractères résultant du remplacement dans *source* de *obsolète* par *nouveau*.

Si *nouveau* est une chaîne vide (""), **Remplacer chaine** supprime chaque occurrence de *obsolète* dans *source*.

Si *combien* est spécifié, **Remplacer chaine** ne remplace que le nombre d'occurrences de *obsolète* spécifié, à partir du premier caractère de *source*. Si *combien* est omis, toutes les occurrences de *obsolète* sont remplacées.

Si *obsolète* est une chaîne vide, **Remplacer chaine** retourne *source* inchangé.

Par défaut, la commande effectue des comparaisons globales, tenant compte des particularités linguistiques et des lettres pouvant s'écrire avec un ou plusieurs caractères (par exemple æ = ae). En revanche, elle n'est pas diacritique (a=A, a=à...) et ne tient pas compte des caractères "ignorables" tels que les caractères dont le code est < 9 (spécification Unicode).
Pour modifier ce fonctionnement, passez l'astérisque * en dernier paramètre. Dans ce cas, les comparaisons sont effectuées sur la base des codes des caractères. Vous devez donc passer le paramètre * :

- si vous souhaitez remplacer des caractères spéciaux, utilisés par exemple comme délimiteurs (**Caractere**(1)...),
- si le remplacement des caractères doit tenir compte de la casse et des accents (a#A, a#à...).
  A noter que dans ce mode, l'évaluation ne gère pas les variations d'écriture des mots.

Note : Dans 4D v15 R3 et suivantes, une optimisation importante a été apportée à l'algorithme utilisé par cette commande lorsque vous remplacez une chaîne par une autre de taille différente, quelle que soit la syntaxe utilisée. Il en résulte une accélération significative des traitements dans ce contexte.

Exemple 1

L'exemple suivant illustre l'utilisation de **Remplacer chaine**. Les résultats sont affectés à la variable *vRésultat*. Les commentaires fournissent la valeur de la variable :

```
vRésultat:=Remplacer chaine("Ville";"ll";"d") ` vRésultat est égal à "Vide"
vRésultat:=Remplacer chaine("Table";"b";"") ` vRésultat est égal à "Tale"
vRésultat:=Remplacer chaine(var;Caractere(Tabulation);",";*) ` Remplacer toutes les tabulations par des virgules
```

Exemple 2

L'exemple suivant élimine les retours chariot et les tabulations du texte contenu dans la variable *vRésultat* :

```
vRésultat:=Remplacer chaine(Remplacer chaine(vRésultat;Caractere(Retour chariot);"";*);Caractere(Tabulation);"";*)
```

Exemple 3

L'exemple suivant illustre le rôle du paramètre * dans le cadre d'une évaluation diacritique :

```
vRésultat:=Remplacer chaine("Crème brûlée";"Brulee";"caramel") `vRésultat est égal à "Crème caramel"
vRésultat:=Remplacer chaine("Crème brûlée";"Brulee";"caramel";*) `vRésultat est égal à "Crème brûlée"
```

Sous chaine

Sous chaine (source ; àPartirDe {; nbCars}) -> Résultat

Paramètre	Type		Description
source	Chaîne	➡	Chaîne de laquelle extraire une sous-chaîne
àPartirDe	Entier long	➡	Position du premier caractère
nbCars	Entier long	➡	Nombre de caractères à extraire
Résultat	Chaîne	⬅	Sous-chaîne de source

Description

La fonction **Sous chaine** retourne la partie de *source* délimitée par les paramètres *àPartirDe* et *nbCars*.

Le paramètre *àPartirDe* indique le premier caractère de la chaîne à retourner, et *nbCars* définit le nombre de caractères à retourner.

Si *nbCars* n'est pas défini ou si le total de *àPartirDe* plus *nbCars* est supérieur au nombre de caractères de la chaîne *source*, **Sous chaine** retourne tous les caractères de la chaîne à partir du caractère spécifié par *àPartirDe*. Si *àPartirDe* est supérieur au nombre de caractères de la chaîne, **Sous chaine** retourne une chaîne vide ("").

Attention : Si vous utilisez cette commande dans un contexte de texte multistyle, il est nécessaire de convertir les éventuels caractères de fin de ligne Windows ('\r\n') en caractères de fin de ligne simples ('\r') afin que les traitements soient valides. Ce principe est lié au mécanisme de normalisation des fins de lignes de 4D assurant la compatibilité de multi-plate-forme des textes. Pour plus d'informations, reportez-vous au paragraphe **Normalisation automatique des fins de lignes**.

Exemple 1

L'exemple suivant illustre l'utilisation de **Sous chaine**. Les résultats sont assignés à la variable *vRésultat*. Les commentaires fournissent la valeur de *vRésultat* :

```
vRésultat:=Sous chaine("08/04/62";4;2) ` vRésultat prend la valeur "04"
vRésultat:=Sous chaine("Important";1;6) ` vRésultat prend la valeur "Import"
vRésultat:=Sous chaine(var;2) ` vRésultat retourne tous les caractères sauf le premier
```

Exemple 2

La méthode projet suivante ajoute au tableau de type texte ou alpha, dont le pointeur est passé en second paramètre, les paragraphes tirés du texte passé en premier paramètre :

```
  ` EXTRAIRE PARAGRAPHES
  ` EXTRAIRE PARAGRAPHES ( Texte ; Pointeur )
  ` EXTRAIRE PARAGRAPHES ( Texte à étudier ; -> Tableau de paragraphes )

C_TEXTE($1)
C_POINTEUR($2)

$vlElem:=Taille tableau($2->)
Repeter
   $vlElem:=$vlElem+1
   INSERER DANS TABLEAU($2->;$vlElem)
   $vlPos:=Position(Caractere(Retour chariot);$1)
   Si($vlPos>0)
      $2->{$vlElem}:=Sous chaine($1;1;$vlPos-1)
      $1:=Sous chaine($1;$vlPos+1)
   Sinon
      $2->{$vlElem}:=$1
   Fin de si
Jusque($1="")
```

Supprimer chaine

Supprimer chaine (source ; positionDépart ; nbCars) -> Résultat

Paramètre	Type		Description
source	Chaîne	➡	Chaîne de départ
positionDépart	Entier long	➡	Premier caractère à supprimer
nbCars	Entier long	➡	Nombre de caractères à supprimer
Résultat	Chaîne	⮌	Chaîne résultante

Description

Supprimer chaine supprime *nbCars* dans *source* à partir de *positionDépart* et retourne la chaîne résultante.

Supprimer chaine retourne la même chaîne que *source* dans les cas suivants :

- *source* est une chaîne vide,
- *positionDépart* est supérieur à la longueur de *source*,
- *nbCars* est égal à zéro (0).

Si *positionDépart* est inférieur à un (1), les caractères sont supprimés à partir du début de la chaîne.

Si *positionDépart* + *nbCars* est supérieur ou égal à la longueur de *source*, les caractères sont supprimés à partir de *positionDépart* jusqu'à la fin de *source*.

Exemple

L'exemple suivant illustre l'utilisation de **Supprimer chaine**. Les résultats sont affectés à la variable *vRésultat*.

```
vRésultat:=Supprimer chaine("Lamborghini";6;6) ` vRésultat est égal à "Lambo"
vRésultat:=Supprimer chaine("Indentation";6;2) ` vRésultat est égal à "Indention"
vRésultat:=Supprimer chaine(var;3;32000) ` vRésultat est égal aux deux premiers caractères de var
```

Trouver regex

Trouver regex (motif ; laChaîne ; début {; pos_trouvée ; long_trouvée}{; *}) -> Résultat

Paramètre	Type		Description
motif	Texte	➡	Expression régulière
laChaîne	Texte	➡	Chaîne dans laquelle s'effectue la recherche
début	Entier long	➡	Position dans laChaîne où doit débuter la recherche
pos_trouvée	Tableau entier long, Variable entier long	⬅	Position de l'occurence
long_trouvée	Tableau entier long, Variable entier long	⬅	Longueur de l'occurence
*	Opérateur	➡	Si passé : rechercher uniquement à la position indiquée
Résultat	Booléen	↩	Vrai = la recherche a trouvé une occurrence, Faux sinon

Trouver regex (motif ; laChaîne) -> Résultat

Paramètre	Type		Description
motif	Texte	➡	Expression régulière (égalité complète)
laChaîne	Texte	➡	Chaîne dans laquelle s'effectue la recherche
Résultat	Booléen	↩	Vrai = la recherche a trouvé une occurrence, Faux sinon

Description

La commande **Trouver regex** permet de tester la conformité d'une chaîne de caractères par rapport à un ensemble de règles synthétisé au moyen d'un méta-langage appelé "expression régulière" ou "expression rationnelle". L'abrévation regex est communément employée pour désigner ces familles de notations.

Passez dans *motif* l'expression régulière à rechercher. Il s'agit d'une suite de caractères chargée de décrire une chaîne de caractères, à l'aide de caractères spéciaux.

Passez dans *laChaîne* la chaîne dans laquelle rechercher l'expression régulière.

Passez dans *début* la position dans *laChaîne* où doit débuter la recherche.

Si *pos_trouvée* et *long_trouvée* sont des variables, la commande retourne la position et la longueur de l'occurrence dans ces variables. Si vous passez des tableaux, la commande retourne la position et la longueur de l'occurrence dans l'élément zéro des tableaux et les positions et longueurs des groupes capturés par l'expression régulière dans les éléments suivants.

Le paramètre * indique, s'il est passé, que la recherche doit s'effectuer à la position définie par *début* sans chercher plus loin en cas d'échec.

La commande retourne **Vrai** si la recherche a trouvé une occurrence.

Pour plus d'informations sur les regex, reportez-vous par exemple à l'adresse suivante :
http://fr.wikipedia.org/wiki/Expression_rationnelle

Pour plus d'informations sur la syntaxe de l'expression régulière passée dans le paramètre *motif*, reportez-vous à l'adresse suivante :
http://www.icu-project.org/userguide/regexp.html

Exemple 1

Recherche d'égalité complète (syntaxe simple) :
vtrouvé:=Trouver regex(motif;montexte)

```
CHERCHER PAR FORMULE([Employés];Trouver regex(".*smith.*";[Employés]nom))
```

Exemple 2

Recherche dans le texte par position :
vtrouvé:=Trouver regex(motif;montexte; début; pos_trouvée; long_trouvée)
Exemple pour afficher tous les tags de $1 :

```
début:=1
Repeter
   vtrouvé:=Trouver regex("<.*>";$1;début;pos_trouvée;long_trouvée)
   Si(vtrouvé)
      ALERTE(Sous chaine($1;pos_trouvée;long_trouvée))
      début:=pos_trouvée+long_trouvée
   Fin de si
Jusque(Non(vtrouvé))
```

Exemple 3

Recherche avec prise en charge des "groupes capturés" via des parenthèses. ( ) permet de définir des groupes dans les regex :
vtrouvé:=Trouver regex(motif;montexte; début; tab_pos_trouvée; tab_long_trouvée)

```
TABLEAU ENTIER LONG(tab_pos_trouvée;0)
TABLEAU ENTIER LONG(tab_long_trouvée;0)
vtrouvé:=Trouver regex("(.*)truc(.*)";$1;1;tab_pos_trouvée;tab_long_trouvée)
Si(vtrouvé)
   $group1:=Sous chaine($1;tab_pos_trouvée{1};tab_long_trouvée{1})
   $group2:=Sous chaine($1;tab_pos_trouvée{2};tab_long_trouvée{2})
Fin de si
```

Exemple 4

Recherche en limitant la comparaison de motif à la position indiquée :
Rajouter une étoile à la fin d'une des deux syntaxes précédentes.

```
vtrouvé:=Trouver regex("a.b";"---a-b---";1;$pos_trouvée;$long_trouvée)
 `retourne Vrai
vtrouvé:=Trouver regex("a.b";"---a-b---";1;$pos_trouvée;$long_trouvée;*)
 `retourne Faux
vtrouvé:=Trouver regex("a.b";"---a-b---";4;$pos_trouvée;$long_trouvée;*)
 `retourne Vrai
```

Note : Les positions et longueurs retournées n'ont de sens qu'en mode Unicode ou si le texte manipulé est de type ASCII 7 bits.

Gestion des erreurs

En cas d'erreur, la commande génère une erreur que vous pouvez intercepter via une méthode installée par la commande **APPELER SUR ERREUR**.

_o_Convertir caractères

_o_Convertir caractères
Ne requiert pas de paramètre

Description

Cette commande est obsolète et ne doit plus être utilisée.

_o_ISO vers Mac

_o_ISO vers Mac (texte) -> Résultat

Paramètre	Type		Description
texte	Chaîne	⇒	Texte en jeu standard Web
Résultat	Chaîne	⇐	Texte en ASCII MacOS

Note de compatibilité

En mode Unicode, cette commande n'a pas d'effet (la chaîne *texte* est retournée sans modification). Depuis la version 11 de 4D, cette commande est obsolète et son usage est déconseillé. Il est recommandé de convertir les chaînes de caractères à l'aide des commandes **CONVERTIR DEPUIS TEXTE** ou **Convertir vers texte**.

_o_Mac vers ISO

_o_Mac vers ISO (texte) -> Résultat

Paramètre	Type		Description
texte	Chaîne	⇒	Texte en ASCII Mac OS
Résultat	Chaîne	⇐	Texte en jeu standard Web

Note de compatibilité

En mode Unicode, cette commande n'a pas d'effet (la chaîne *texte* est retournée sans modification). Depuis la version 11 de 4D, cette commande est obsolète et son usage est déconseillé. Il est recommandé de convertir les chaînes de caractères à l'aide des commandes **CONVERTIR DEPUIS TEXTE** ou **Convertir vers texte**.

_o_Mac vers Windows

_o_Mac vers Windows (texte) -> Résultat

Paramètre	Type		Description
texte	Chaîne	➡	Texte exprimé en ASCII Mac OS
Résultat	Chaîne	⮌	Texte exprimé en ANSI Windows

Note de compatibilité

En mode Unicode, cette commande n'a pas d'effet (la chaîne *texte* est retournée sans modification). Depuis la version 11 de 4D, cette commande est obsolète et son usage est déconseillé. Il est recommandé de convertir les chaînes de caractères à l'aide des commandes **CONVERTIR DEPUIS TEXTE** ou **Convertir vers texte**.

_o_Windows vers Mac

_o_Windows vers Mac (texte) -> Résultat

Paramètre	Type		Description
texte	Chaîne	➡	Texte en ANSI Windows
Résultat	Chaîne	⮌	Texte en ASCII Mac OS

Note de compatibilité

En mode Unicode, cette commande n'a pas d'effet (la chaîne *texte* est retournée sans modification). Depuis la version 11 de 4D, cette commande est obsolète et son usage est déconseillé. Il est recommandé de convertir les chaînes de caractères à l'aide des commandes **CONVERTIR DEPUIS TEXTE** ou **Convertir vers texte**.

Client HTTP

HTTP AUTHENTIFIER
HTTP FIXER DOSSIER CERTIFICATS
HTTP FIXER OPTION
HTTP Get
HTTP Lire dossier certificats
HTTP LIRE OPTION
HTTP Request

HTTP AUTHENTIFIER

HTTP AUTHENTIFIER (nom ; motDePasse {; méthodeAuth} {; *})

Paramètre	Type		Description
nom	Texte	⇒	Nom de l'utilisateur
motDePasse	Texte	⇒	Mot de passe de l'utilisateur
méthodeAuth	Entier long	⇒	Méthode d'authentification : 0 ou omis=non définie, 1=BASIC, 2=DIGEST
*	Opérateur	⇒	Si passé : authentification par proxy

Description

La commande **HTTP AUTHENTIFIER** vous permet d'effectuer des requêtes HTTP vers des serveurs nécessitant l'authentification de l'application cliente. Les méthodes BASIC et DIGEST sont prises en charge ainsi que la présence d'un proxy.

Passez dans les paramètres *nom* et *motDePasse* les informations d'identification requises (nom d'utilisateur et mot de passe). Ces informations seront encodées et ajoutées à la prochaine requête HTTP envoyée via la commande **HTTP Request** ou **HTTP Get**. Il est donc nécessaire d'appeler la commande **HTTP AUTHENTIFIER** avant chaque requête HTTP.

Le paramètre facultatif *méthodeAuth* permet d'indiquer la méthode d'authentification à utiliser. Vous pouvez passer l'une des constantes suivantes, placées dans le thème **Client HTTP** :

Constante	Type	Valeur	Comment
HTTP basic	Entier long	1	Utiliser la méthode d'authentification BASIC
HTTP digest	Entier long	2	Utiliser la méthode d'authentification DIGEST

Si vous omettez le paramètre *méthodeAuth* (ou passez 0), vous laissez le programme utiliser la méthode appropriée. Dans ce cas, 4D envoie une requête supplémentaire afin de négocier la méthode d'authentification.

Si vous passez le paramètre *, vous indiquez que les informations d'authentification s'adressent à un proxy HTTP. Ce paramétrage doit être mis en oeuvre lorsqu'il existe un proxy nécessitant une authentification entre le client et le serveur HTTP. Si le serveur est lui-même authentifié, une double authentification est requise.

Par défaut, les informations d'authentification sont conservées et réutilisées dans le process courant. Vous pouvez toutefois les effacer après chaque requête à l'aide d'une option de la commande **HTTP FIXER OPTION**. Dans ce cas, il sera nécessaire d'exécuter la commande **HTTP AUTHENTIFIER** avant tout appel à **HTTP Request** ou **HTTP Get**.

Exemple

Exemples de requêtes avec authentification :

```
 //Authentification sur un serveur HTTP en mode DIGEST
HTTP AUTHENTIFIER("httpUser";"123";2)
 //Authentification sur un proxy en mode par défaut
HTTP AUTHENTIFIER("ProxyUser";"456";*)
$httpStatus:=HTTP Get(...)
```

HTTP FIXER DOSSIER CERTIFICATS

HTTP FIXER DOSSIER CERTIFICATS (dossierCertificats)

Paramètre	Type		Description
dossierCertificats	Texte	➡	Chemin d'accès et nom du dossier des certificats du client

Description

La commande **HTTP FIXER DOSSIER CERTIFICATS** permet de modifier le dossier de certificats client actif pour l'ensemble des process dans la session courante.

Le dossier de certificats client est celui dans lequel 4D va rechercher les fichiers des certificats clients réclamés par les serveurs Web. Par défaut, tant que la commande **HTTP FIXER DOSSIER CERTIFICATS** n'est pas exécutée, 4D utilise un dossier nommé "ClientCertificatesFolder" créé à côté du fichier de structure. Ce dossier n'est créé que si nécessaire.

La possibilité d'utiliser plusieurs certificats clients est une nouveauté de 4D v14.

Passez dans *dossierCertificats* le chemin d'accès du dossier personnalisé contenant les certificats clients. Vous pouvez passer soit un chemin d'accès relatif au fichier de structure de l'application, soit un chemin d'accès absolu. Le chemin doit être exprimé avec la syntaxe système, par exemple :

- (OS X) : Disk:Applications:myserv:folder
- (Windows) : C:\Applications\myserv\folder

Lorsque cette commande a été exécutée, le nouveau chemin est immédiatement pris en compte par les commandes telles que **HTTP Request** exécutées ultérieurement (il n'est pas nécessaire de redémarrer l'application). Il est utilisé dans tous les process de la base.

Si le dossier spécifié n'existe pas à l'emplacement défini ou si le chemin d'accès passé dans *dossierCertificats* est invalide, une erreur est générée, que vous pouvez intercepter à l'aide d'une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

A propos des certificats SSL

Comme décrit dans la section **Utiliser le protocole TLS**, les certificats SSL gérés par 4D doivent être au **format PEM**. Si vous récupérez auprès de votre fournisseur de certificat (par exemple startssl) un certificat dans un format binaire tel que .crt, .pfx ou .p12 (le format dépend également de votre navigateur), vous devrez le convertir au format PEM pour pouvoir l'utiliser. Des sites Web tels que sslshopper vous permettront d'effectuer la conversion en ligne.

Exemple

Vous souhaitez changer temporairement de dossier de certificats :

```
C_TEXTE($certifFolder)
$certifFolder :=HTTP Lire dossier certificats //on stocke le dossier courant
HTTP FIXER DOSSIER CERTIFICATS("C:/temp/certifTempo/")
   ... // exécution de requêtes spécifiques
HTTP FIXER DOSSIER CERTIFICATS($certifFolder) //on rétablit le dossier
```

HTTP FIXER OPTION

HTTP FIXER OPTION (option ; valeur)

Paramètre	Type		Description
option	Entier long	➡	Code de l'option à fixer
valeur	Entier long	➡	Valeur de l'option

Description

La commande **HTTP FIXER OPTION** permet de définir différentes options qui seront utilisées lors de la prochaine requête HTTP déclenchée par les commandes **HTTP Get** ou **HTTP Request**. Vous pouvez appeler cette commande autant de fois qu'il y a d'options à fixer.

Note : Les options définies sont locales au process courant. Dans le cadre d'un composant, elles sont locales au composant en cours d'exécution.

Passez dans le paramètre *option* le numéro de l'option à définir et dans le paramètre valeur la nouvelle *valeur* de l'option. Vous pouvez utiliser pour le paramètre *option* une des constantes prédéfinies suivantes, placées dans le thème **Client HTTP** :

Constante	Type	Valeur	Comment
HTTP afficher dial auth	Entier long	4	*valeur* = 0 (ne pas afficher le dialogue) ou 1 (afficher le dialogue). Par défaut : 0 Cette option gère l'affichage de boîte de dialogue d'authentification lors de l'exécution de la commande **HTTP Get** ou **HTTP Request**. Par défaut, cette commande ne provoque jamais l'affichage de la boîte de dialogue, vous devez en principe utiliser la commande **HTTP AUTHENTIFIER**. Toutefois, si vous souhaitez qu'une boîte de dialogue d'authentification apparaisse pour que l'utilisateur saisisse ses identifiants, passez 1 dans valeur. La boîte de dialogue n'apparaît que si la requête requiert une authentification.
HTTP compression	Entier long	6	*valeur* = 0 (ne pas compresser) ou 1 (compresser). Par défaut : 0 Cette option permet d'activer ou d'activer le mécanisme de compression des requêtes entre le client et le serveur, destiné à accélérer les échanges. Lorsque ce mécanisme est activé, le client HTTP utilise la compression deflate ou GZIP en fonction de la réponse du serveur.
HTTP effacer infos auth	Entier long	5	*valeur* = 0 (ne pas effacer les informations) ou 1 (les effacer). Par défaut : 0 Cette option permet d'indiquer à 4D de réinitialiser les informations d'authentification de l'utilisateur (nom d'utilisateur, mot de passe, méthode) après chaque exécution d'une commande **HTTP Get** ou **HTTP Request** dans un même process. Par défaut, ces informations sont conservées et réutilisées à chaque requête. Passez 1 dans *valeur* pour les effacer après chaque appel. A noter que quel que soit le paramétrage, les informations sont effacées lorsque le process est détruit.
HTTP redirections max	Entier long	3	*valeur* = nombre maximum de redirections acceptées Valeur par défaut = 2
HTTP suivre redirection	Entier long	2	*valeur* = 0 (ne pas accepter les redirections) ou 1 (accepter les redirections). Valeur par défaut = 1
HTTP timeout	Entier long	1	*valeur* = timeout de la requête cliente, exprimé en secondes. Le timeout est le délai d'attente du client HTTP en cas de non-réponse du serveur. A l'issue de ce délai, le client referme la session, la requête est perdue. Par défaut, ce délai est de 120 secondes. Il peut être modifié en raison de caractéristiques particulières (état du réseau, spécificités de la requête, etc.).

L'ordre d'appel des options n'a pas d'importance. Si une même option est définie plusieurs fois, seule la valeur du dernier appel est prise en compte.

HTTP Get

HTTP Get (url ; réponse {; nomsEnTêtes ; valeursEnTêtes}{; *}) -> Résultat

Paramètre	Type		Description
url	Texte	➡	URL auquel envoyer la requête
réponse	Texte, BLOB, Image, Objet	⬅	Résultat de la requête
nomsEnTêtes	Tableau texte	➡	Noms des en-têtes de la requête
		⬅	Noms d'en-têtes retournés
valeursEnTêtes	Tableau texte	➡	Valeurs d'en-têtes de la requête
		⬅	Valeurs d'en-têtes retournées
*	Opérateur	➡	Si passé, la connexion est maintenue (keep-alive) Si omis, la connexion est automatiquement refermée
Résultat	Entier long	↩	Code de statut HTTP

Description

La commande **HTTP Get** permet d'envoyer directement une requête HTTP GET vers un URL spécifique et de traiter la réponse du serveur HTTP.

Passez dans le paramètre *url* l'URL auquel adresser la requête. La syntaxe à utiliser est :

```
http://[{user}:[{password}]@]host[:{port}][/{path}][?{queryString}]
```

Par exemple, les chaînes suivantes peuvent être passées :

```
http://www.myserver.com
http://www.myserver.com/path
http://www.myserver.com/path?name="jones"
https://www.myserver.com/login (*)
http://123.45.67.89:8083
http://john:smith@123.45.67.89:8083
```

(*) Lors des requêtes https, l'autorité du certificat n'est pas vérifiée.

Après exécution de la commande, le paramètre *réponse* récupère le résultat de la requête retourné par le serveur. Ce résultat correspond à la partie corps (*body*) de la réponse, sans les en-têtes (*headers*). Vous pouvez passer des variables de différents types dans *réponse* :

- Texte : lorsque le résultat est attendu sous forme de texte (cf. note)
- BLOB : lorsque le résultat est attendu sous forme binaire
- Image : lorsque le résultat est attendu sous forme d'image
- Objet : lorsque le résultat est attendu sous forme d'objet **C_OBJET**

Note : Lorsqu'une variable texte est passée dans *réponse*, 4D tente de décoder les données retournées par le serveur. Le programme essaie d'abord de récupérer le charset depuis l'en-tête *content-type*, ou à défaut via la BOM de la page ; en dernier lieu 4D recherche tout attribut *http-equiv charset* (dans le contenu html) ou *encoding* (pour le xml). Si aucun charset ne peut être détecté, 4D décode la réponse en ANSI. Si la conversion échoue, le texte résultant est vide. Si vous n'êtes pas sûr que le serveur retourne une information de charset ou une BOM, mais si vous connaissez l'encodage, il est préférable de passer un BLOB dans *réponse* et d'utiliser la commande **Convertir vers texte**.

Si vous passez un BLOB, il contiendra le texte, l'image ou tout type de contenu (.wav, .zip...) retourné par le serveur. Vous devrez alors gérer la récupération de ce contenu (les en-têtes ne sont pas inclus dans le BLOB).
Si vous passez un objet de type **C_OBJET** et si la requête retourne un résultat ayant le content-type "application/json" (ou "*quelquechose*/json"), 4D tentera d'analyser le contenu JSON afin de générer l'objet.

Vous pouvez passer dans les paramètres *nomsEnTêtes* et *valeursEnTêtes* des tableaux contenant respectivement les noms et les valeurs des en-têtes de la requête.
A l'issue de l'exécution de la méthode, ces tableaux contiendront les noms et valeurs d'en-têtes retournés par le serveur HTTP. Ce principe permet notamment de gérer des cookies.

Le paramètre * permet d'activer le mécanisme de *keep-alive* pour la connexion au serveur. Par défaut, si ce paramètre est omis, le *keep-alive* n'est pas activé.

La commande retourne le code de statut HTTP standard (200=OK...) tel que renvoyé par le serveur. La liste des codes de statut HTTP est fournie dans la RFC 2616. Si la connexion au serveur est impossible pour une raison liée au réseau (*DNS Failed*, *Server not reachable*...) la commande retourne 0 et une erreur est générée. Vous pouvez intercepter les erreurs à l'aide d'une méthode d'appel sur erreur installée par la commande **APPELER SUR ERREUR**.

Exemple 1

Récupération du logo 4D sur le site Web de 4D :

```
C_TEXTE(URLPic_t)
URLPic_t:="http://www.4d.com/sites/all/themes/dimention/images/home/logo4D.jpg"
TABLEAU TEXTE(HeaderNames_at;0)
TABLEAU TEXTE(HeaderValues_at;0)
C_IMAGE(Pic_i)
$httpResponse:=HTTP Get(URLPic_t;Pic_i;HeaderNames_at;HeaderValues_at)
```

Exemple 2

Récupération d'une RFC :

```
C_TEXTE(URLText_t)
C_TEXTE(Text_t)
URLText_t:="http://tools.ietf.org/rfc/rfc1.txt"
TABLEAU TEXTE(HeaderNames_at;0)
TABLEAU TEXTE(HeaderValues_at;0)
$httpResponse:=HTTP Get(URLText_t;Text_t;HeaderNames_at;HeaderValues_at)
```

Exemple 3

Récupération d'une vidéo :

```
C_BLOB(vBlob)
$httpResponse:=HTTP Get("http://www.example.com/video.flv";vBlob)
BLOB VERS DOCUMENT("video.flv";vBlob)
```

HTTP Lire dossier certificats

HTTP Lire dossier certificats -> Résultat

Paramètre	Type	Description
Résultat	Texte	Chemin d'accès complet du dossier de certificats actif

Description

La commande **HTTP Lire dossier certificats** retourne le chemin d'accès complet du dossier de certificats client actif.

Par défaut, 4D utilise le dossier "ClientCertificatesFolder" créé à côté du fichier de structure (dossier créé uniquement si nécessaire). Vous pouvez toutefois créer un dossier personnalisé pour le process courant à l'aide de la commande **HTTP FIXER DOSSIER CERTIFICATS**.

Exemple

Vous souhaitez changer temporairement de dossier de certificats :

```
C_TEXTE($certifFolder)
$certifFolder :=HTTP Lire dossier certificats //on stocke le dossier courant
HTTP FIXER DOSSIER CERTIFICATS("C:/temp/certifTempo/")
   ... // exécution de requêtes spécifiques
HTTP FIXER DOSSIER CERTIFICATS($certifFolder) //on rétablit le dossier
```

HTTP LIRE OPTION

HTTP LIRE OPTION (option ; valeur)

Paramètre	Type		Description
option	Entier long	⇒	Code de l'option à lire
valeur	Entier long	⇐	Valeur courante de l'option

Description

La commande **HTTP LIRE OPTION** retourne la valeur courante des options HTTP (options utilisées par le client pour la prochaine requête déclenchée par la commande **HTTP Get** ou **HTTP Request**). La valeur courante d'une option peut être la valeur par défaut ou avoir été modifiée à l'aide de la commande **HTTP FIXER OPTION**.

Note : Les options sont locales au process courant. Dans le cadre d'un composant, elles sont locales au composant en cours d'exécution.

Passez dans le paramètre *option* le numéro de l'option dont vous souhaitez lire la valeur. Vous pouvez utiliser une des constantes prédéfinies suivantes, placées dans le thème **Client HTTP** :

Constante	Type	Valeur	Comment
HTTP afficher dial auth	Entier long	4	*valeur* = 0 (ne pas afficher le dialogue) ou 1 (afficher le dialogue). Par défaut : 0 Cette option gère l'affichage de boîte de dialogue d'authentification lors de l'exécution de la commande **HTTP Get** ou **HTTP Request**. Par défaut, cette commande ne provoque jamais l'affichage de la boîte de dialogue, vous devez en principe utiliser la commande **HTTP AUTHENTIFIER**. Toutefois, si vous souhaitez qu'une boîte de dialogue d'authentification apparaisse pour que l'utilisateur saisisse ses identifiants, passez 1 dans valeur. La boîte de dialogue n'apparaît que si la requête requiert une authentification.
HTTP compression	Entier long	6	*valeur* = 0 (ne pas compresser) ou 1 (compresser). Par défaut : 0 Cette option permet d'activer ou d'activer le mécanisme de compression des requêtes entre le client et le serveur, destiné à accélérer les échanges. Lorsque ce mécanisme est activé, le client HTTP utilise la compression deflate ou GZIP en fonction de la réponse du serveur.
HTTP effacer infos auth	Entier long	5	*valeur* = 0 (ne pas effacer les informations) ou 1 (les effacer). Par défaut : 0 Cette option permet d'indiquer à 4D de réinitialiser les informations d'authentification de l'utilisateur (nom d'utilisateur, mot de passe, méthode) après chaque exécution d'une commande **HTTP Get** ou **HTTP Request** dans un même process. Par défaut, ces informations sont conservées et réutilisées à chaque requête. Passez 1 dans *valeur* pour les effacer après chaque appel. A noter que quel que soit le paramétrage, les informations sont effacées lorsque le process est détruit.
HTTP redirections max	Entier long	3	*valeur* = nombre maximum de redirections acceptées Valeur par défaut = 2
HTTP suivre redirection	Entier long	2	*valeur* = 0 (ne pas accepter les redirections) ou 1 (accepter les redirections). Valeur par défaut = 1
HTTP timeout	Entier long	1	*valeur* = timeout de la requête cliente, exprimé en secondes. Le timeout est le délai d'attente du client HTTP en cas de non-réponse du serveur. A l'issue de ce délai, le client referme la session, la requête est perdue. Par défaut, ce délai est de 120 secondes. Il peut être modifié en raison de caractéristiques particulières (état du réseau, spécificités de la requête, etc.).

Passez dans le paramètre *valeur* une variable qui recevra la valeur courante de l'*option*.

HTTP Request

HTTP Request (méthodeHTTP ; url ; contenu ; réponse {; nomsEnTêtes ; valeursEnTêtes}{; *}) -> Résultat

Paramètre	Type		Description
méthodeHTTP	Texte	➡	Méthode HTTP pour la requête
url	Texte	➡	URL auquel envoyer la requête
contenu	Texte, BLOB, Image, Objet	➡	Contenu du corps (body) de la requête
réponse	Texte, BLOB, Image, Objet	⬅	Résultat de la requête
nomsEnTêtes	Tableau texte	➡	Noms des en-têtes de la requête
		⬅	Noms d'en-têtes retournés
valeursEnTêtes	Tableau texte	➡	Valeurs d'en-têtes de la requête
		⬅	Valeurs d'en-têtes retournées
*	Opérateur	➡	Si passé, la connexion est maintenue (keep-alive) Si omis, la connexion est automatiquement refermée
Résultat	Entier long	➡	Code de statut HTTP

Description

La commande **HTTP Request** permet d'envoyer tout type de requête HTTP vers un URL spécifique et de traiter la réponse du serveur HTTP.

Passez dans le paramètre *méthodeHTTP* la méthode HTTP de la requête. Vous pouvez utiliser une des constantes suivantes, placées dans le thème **Client HTTP** :

Constante	Type	Valeur	Comment
HTTP méthode DELETE	Chaîne	DELETE	Voir la RFC 2616
HTTP méthode GET	Chaîne	GET	Voir la RFC 2616. Equivaut à utiliser la commande **HTTP Get**
HTTP méthode HEAD	Chaîne	HEAD	Voir la RFC 2616
HTTP méthode OPTIONS	Chaîne	OPTIONS	Voir la RFC 2616
HTTP méthode POST	Chaîne	POST	Voir la RFC 2616
HTTP méthode PUT	Chaîne	PUT	Voir la RFC 2616
HTTP méthode TRACE	Chaîne	TRACE	Voir la RFC 2616

Passez dans le paramètre *url* l'URL auquel adresser la requête. La syntaxe à utiliser est :

```
http://[{user}:[{password}]@]host[:{port}][/{path}][?{queryString}]
```

Par exemple, les chaînes suivantes peuvent être passées :

```
http://www.myserver.com
http://www.myserver.com/path
http://www.myserver.com/path?name="jones"
https://www.myserver.com/login (*)
http://123.45.67.89:8083
http://john:smith@123.45.67.89:8083
```

(*) Lors des requêtes https, l'autorité du certificat n'est pas vérifiée.

Passez dans le paramètre *contenu* le corps (*body*) de la requête. Les données à passer dans ce paramètre dépendent de la méthode HTTP de la requête. Vous pouvez envoyer des données de type texte, BLOB, image ou objet. Lorsque le *content-type* n'est pas spécifié, les types suivants sont utilisés :

- pour les textes : text/plain - UTF8
- pour les BLOB : application/octet-stream
- pour les images : type mime connu (*best for Web*)
- pour les objets **C_OBJET** : application/json

Après exécution de la commande, le paramètre *réponse* récupère le résultat de la requête retourné par le serveur. Ce résultat correspond à la partie "corps" (*body*) de la réponse, sans les "en-têtes" (*headers*). Vous pouvez passer des variables de différents types dans *réponse* :

- Texte : lorsque le résultat est attendu sous forme de texte (cf. note ci-dessous).
- BLOB : lorsque le résultat est attendu sous forme binaire.
- Image : lorsque le résultat est attendu sous forme d'image.
- Objet **C_OBJET** : lorsque le résultat est attendu sous forme d'objet.

Note : Lorsqu'une variable texte est passée dans *réponse*, 4D tente de décoder les données retournées par le serveur. Le programme essaie d'abord de récupérer le charset depuis l'en-tête *content-type*, ou à défaut via la BOM de la page ; en dernier lieu 4D recherche tout attribut *http-equiv charset* (dans le contenu html) ou *encoding* (pour le xml). Si aucun charset ne peut être détecté, 4D décode la réponse en ANSI. Si la conversion échoue, le texte résultant est vide. Si vous n'êtes pas sûr que le serveur retourne une information de charset ou une BOM, mais si vous connaissez l'encodage, il est préférable de passer un BLOB dans *réponse* et d'utiliser la commande **Convertir vers texte**.

Si vous passez une variable de type **C_OBJET** dans le paramètre *réponse* et si la requête retourne un résultat ayant le content-type "application/json" (ou "*quelquechose*/json"), 4D tentera d'analyser le contenu JSON afin de générer l'objet.

Si le résultat retourné par le serveur ne correspond pas au type de la variable *réponse*, elle est laissée vide et la variable système OK prend la valeur 0.

Vous pouvez passer dans les paramètres *nomsEnTêtes* et *valeursEnTêtes* des tableaux contenant respectivement les noms et les valeurs des en-têtes de la requête.
A l'issue de l'exécution de la méthode, ces tableaux contiendront les noms et valeurs des en-têtes retournés par le serveur HTTP. Ce principe permet notamment de gérer des cookies.

Le paramètre * permet d'activer le mécanisme de *keep-alive* pour la connexion au serveur. Par défaut, si ce paramètre est omis, le *keep-alive* n'est pas activé.

La commande retourne le code de statut HTTP standard (200=OK...) tel que renvoyé par le serveur. La liste des codes de statut HTTP est fournie dans la

RFC 2616.
Si la connexion au serveur est impossible pour une raison liée au réseau (*DNS Failed*, *Server not reachable*...) la commande retourne 0 et une erreur est générée. Vous pouvez intercepter les erreurs à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

Exemple 1

Demande de suppression d'un enregistrement dans une base distante :

```
C_TEXTE($response)
$body_t:="{record_id:25}"
$httpStatus_l:=HTTP Request(HTTP méthode DELETE;"database.example.com";$body_t;$response)
```

Note : Il vous appartient de traiter la demande de manière appropriée au niveau du serveur distant, la commande **HTTP Request** gère uniquement la requête et le résultat retourné.

Exemple 2

Demande d'ajout d'un enregistrement dans une base distante :

```
C_TEXTE($response)
$body_t:="{fName:'john',fName:'Doe'}"
$httpStatus_l:=HTTP Request(HTTP méthode PUT;"database.example.com";$body_t;$response)
```

Note : Il vous appartient de traiter la demande de manière appropriée au niveau du serveur distant, la commande **HTTP Request** gère uniquement la requête et le résultat retourné.

Exemple 3

Demande d'ajout d'enregistrement en JSON dans une base distante :

```
C_OBJET($content)
OB FIXER($content;"nom";"Doe";"prénom";"John")
$result:=HTTP Request(HTTP méthode PUT;"database.example.com";$content;$response)
```

Communications

- ENVOYER ENREGISTREMENT
- ENVOYER PAQUET
- ENVOYER VARIABLE
- FIXER TIMEOUT
- LIRE CORRESPONDANCE PORT SERIE
- RECEVOIR BUFFER
- RECEVOIR ENREGISTREMENT
- RECEVOIR PAQUET
- RECEVOIR VARIABLE
- REGLER SERIE
- UTILISER FILTRE

ENVOYER ENREGISTREMENT

ENVOYER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle envoyer l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

ENVOYER ENREGISTREMENT envoie l'enregistrement courant de *laTable* vers le port série ou vers un document ouvert par la commande **REGLER SERIE**. L'enregistrement est envoyé dans un format interne particulier ne pouvant être interprété que par la commande **RECEVOIR ENREGISTREMENT**. S'il n'y a pas d'enregistrement courant, **ENVOYER ENREGISTREMENT**ne fait rien.

L'enregistrement est envoyé en totalité, ce qui signifie que les images et les BLOBs stockés dans ou avec l'enregistrement sont également envoyés.

Important : Lorsque des enregistrements sont envoyés et reçus par **ENVOYER ENREGISTREMENT** et **RECEVOIR ENREGISTREMENT**, la structure de la table source et celle de la table de destination doivent être compatibles. Si ce n'est pas le cas, 4D convertira les valeurs en fonction des définitions des tables lorsque **RECEVOIR ENREGISTREMENT** sera exécutée.

Note : Si vous envoyez un enregistrement à un document avec cette commande, le document doit avoir été ouvert par la commande **REGLER SERIE**. Vous ne pouvez pas utiliser **ENVOYER ENREGISTREMENT** avec un document ouvert par **Ouvrir document**, **Creer document** ou **Ajouter a document**.

Note de compatibilité : A compter de la version 11 de 4D, cette commande ne prend plus en charge les sous-tables.

Exemple

Reportez-vous à l'exemple de la commande **RECEVOIR ENREGISTREMENT**.

ENVOYER PAQUET

ENVOYER PAQUET ({docRef ;} paquet)

Paramètre	Type		Description
docRef	RefDoc	⇒	Référence de document ou canal courant (port série ou document)
paquet	Chaîne, BLOB	⇒	Chaîne ou BLOB à envoyer

Description

La commande **ENVOYER PAQUET** envoie *paquet* vers un port série ou un document. Si *docRef* est spécifié, le paquet est écrit dans le document référencé par *docRef*. Si *docRef* n'est pas spécifié, le paquet est envoyé vers le port série ou un document préalablement ouvert par la commande **REGLER SERIE**.

paquet représente une simple série de données, généralement une chaîne de caractères.
Vous pouvez également passer un BLOB dans *paquet*. Ce principe permet notamment de s'affranchir des contraintes liées à l'encodage des caractères envoyés en mode texte (cf. exemple 2).

Note : Lorsque vous passez un BLOB dans *paquet*, la commande ne tient pas compte du jeu de caractères éventuellement défini par la commande **UTILISER FILTRE**. Le BLOB est envoyé sans aucune modification.

Avant d'utiliser **ENVOYER PAQUET**, vous devez ouvrir un port série ou un document avec la commande **REGLER SERIE**, ou un document avec une commande de gestion des documents.

Lorsque vous envoyez un paquet vers un document, le premier **ENVOYER PAQUET** commence à écrire les données au début du document — à moins que ce dernier n'ait été ouvert par la fonction **UTILISER FILTRE**. Puis, jusqu'à ce que le document soit refermé, chaque paquet envoyé y est écrit à la suite du précédent.

Note : Ce fonctionnement est valide avec un document ouvert par **REGLER SERIE**. Cependant, pour un document ouvert par **Ouvrir document**, **Creer document** ou **Ajouter a document**, vous pouvez utiliser les commandes **Position dans document** et **CHANGER POSITION DANS DOCUMENT** pour connaître et modifier la position à laquelle, dans le document, la prochaine écriture (**ENVOYER PAQUET**) ou lecture (**RECEVOIR PAQUET**) aura lieu.

Exemple 1

L'exemple suivant écrit, dans un document, des données en provenance de champs. Les valeurs sont écrites sous forme de champs de taille fixe. Dans ce cas, si la longueur d'un champ est inférieure à la taille fixée, le champ est comblé avec des espaces (c'est-à-dire que des espaces sont ajoutés de manière à ce que le champ corresponde à la taille définie). Bien que les champs de valeurs fixes soient un moyen peu efficace de stocker des données, certains systèmes informatiques et certaines applications l'utilisent encore :

```
$Doc :=Creer document("") ` Création d'un document
Si(OK=1) ` Est-ce que le document a bien été créé ?
   Boucle($i;1;Enregistrements trouves([Personnes])) ` Boucle pour chaque enregistrement
      ENVOYER PAQUET($Doc;Remplacer caracteres(15*Caractere(Espacement);[Personnes]Prénom;1))
   ` Envoi du paquet créé à partir d'une chaîne de 15 espaces contenant le champ Prénom.
      ENVOYER PAQUET($Doc;Remplacer caracteres(15*Caractere(Espacement);[Personnes]Nom;1))
   ` Envoi d'un second paquet créé à partir d'une chaîne de 15 espaces contenant le champ Nom.
   ` Cela aurait pu être mis dans le premier paquet, mais est séparé pour des raisons de clarté.
      ENREGISTREMENT SUIVANT([Personnes])
   Fin de boucle
   ENVOYER PAQUET($Doc;Caractere(ASCII SUB))
   ` Envoi du code ASCII SUB, utilisé comme marqueur de fin d'enregistrement par certains ordinateurs.
   FERMER DOCUMENT($Doc) ` Fermeture du document
Fin de si
```

Exemple 2

Cet exemple illustre l'envoi et la récupération de caractères étendus via un BLOB dans un document :

```
C_BLOB($blob_envoi)
C_BLOB($blob_reception)
TEXTE VERS BLOB("âzérty";$blob_envoi;UTF8 texte sans longueur)
FIXER TAILLE BLOB($blob_envoi;16;255)
$blob_envoi{6}:=0
$blob_envoi{7}:=1
$blob_envoi{8}:=2
$blob_envoi{9}:=3
$blob_envoi{10}:=0
$vlDocRef:=Creer document("blob.test")
Si(OK=1)
   ENVOYER PAQUET($vlDocRef;$blob_envoi)
   FERMER DOCUMENT($vlDocRef)
Fin de si
$vlDocRef:=Ouvrir document(document)
Si(OK=1)
   RECEVOIR PAQUET($vlDocRef;$blob_reception;65536)
   FERMER DOCUMENT($vlDocRef)
Fin de si
```

ENVOYER VARIABLE

ENVOYER VARIABLE (variable)

Paramètre	Type		Description
variable	Variable	⇨	Variable à envoyer

Description

ENVOYER VARIABLE envoie *variable* vers le document ou le port série préalablement ouvert par la commande **REGLER SERIE**. La variable est envoyée dans un format interne spécial qui ne peut être relu que par la commande **RECEVOIR VARIABLE**. **ENVOYER VARIABLE** envoie la totalité de la variable (y compris son type et sa valeur).

Notes :

1. Si vous envoyez une variable à un document avec cette commande, le document doit avoir été ouvert par la commande **REGLER SERIE**. Vous ne pouvez pas utiliser **ENVOYER VARIABLE** avec un document ouvert par **Ouvrir document**, **Creer document** ou **Ajouter a document**.
2. Cette commande ne fonctionne pas avec les variables de type tableau. Si vous voulez envoyer et recevoir des tableaux via un document ou un port série, utilisez les **Commandes du thème BLOB**.

Exemple

Reportez-vous à l'exemple de la commande **RECEVOIR ENREGISTREMENT**.

FIXER TIMEOUT

FIXER TIMEOUT (secondes)

Paramètre	Type		Description
secondes	Entier long	⇨	Nombre de secondes jusqu'au timeout

Description

La commande **FIXER TIMEOUT** vous permet de définir le temps d'attente maximum pour l'exécution d'une commande de communication série. Si la commande ne se termine pas dans le temps *secondes* qui lui est imparti, la communication série est annulée, l'erreur –9990 est générée, et la variable système OK prend la valeur 0. Vous pouvez intercepter cette erreur à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

Notez que le délai défini représente la durée totale permise pour que la commande s'exécute, et non le délai d'attente entre chaque caractère reçu. Pour annuler un paramétrage précédent et ne pas spécifier de temps d'attente maximum, passez 0 dans le paramètre *secondes*.

Les commandes de communication série affectées par ce paramétrage sont les suivantes :

- **RECEVOIR PAQUET**
- **RECEVOIR ENREGISTREMENT**
- **RECEVOIR VARIABLE**

Exemple

L'exemple suivant fixe le port série devant recevoir des données et le timeout. Les données sont lues à l'aide de **RECEVOIR PAQUET**. Si les données ne sont pas bien reçues dans le temps défini, une erreur survient :

```
  ` Ouverture du port série
REGLER SERIE(Port série MacOS;Vitesse 9600+Bits de données 8+Bit de stop un+Pas de parité)
FIXER TIMEOUT(10) ` Fixer le timeout à 10 secondes
APPELER SUR ERREUR("INTERCEPTER ERREURS COMMUNICATIONS") ` Traiter les interruptions éventuelles
RECEVOIR PAQUET(vBuffer;Caractere(Retour chariot)) ` Lire jusqu'au retour chariot
Si(OK=0) ` Si une erreur survient
   ALERTE("Erreur lors de la réception des données.") ` Informer l'utilisateur
Sinon
   [Personnes]Nom:=vBuffer ` Sauvegarder les données dans un champ
Fin de si
```

LIRE CORRESPONDANCE PORT SERIE

LIRE CORRESPONDANCE PORT SERIE (tabNums ; tabLibellés)

Paramètre	Type		Description
tabNums	Tableau entier long	⇐	Tableau de numéros de ports série
tabLibellés	Tableau chaîne	⇐	Tableau de noms de ports série

Description

La commande **LIRE CORRESPONDANCE PORT SERIE** retourne deux tableaux *tabNums* et *tabLibellés* contenant respectivement la liste des numéros et des noms des ports série de la machine courante.

Cette commande est utile sous Mac OS X car le système alloue dynamiquement les numéros des ports série lorsque vous utilisez un adaptateur série USB. A l'aide de cette commande, vous pouvez adresser les ports série étendus via leur nom (invariable), quel que soit leur numéro.

Note : Cette commande ne retourne pas de valeurs significatives avec les ports standard. Si vous souhaitez adresser un port standard, vous devez passer directement sa valeur (0 ou 1) à la commande **REGLER SERIE** (ancien mode de fonctionnement de 4D).

Exemple

Cette méthode projet permet d'adresser le même port série (sans protocole), quel que soit le numéro qui lui a été attribué :

```
TABLEAU TEXTE($tNomPorts;0)
TABLEAU ENTIER LONG($tNumPorts;0)
C_ENTIER LONG($vNumport;$vNumportFinal)

 `Connaître les numéros actuels des ports série
LIRE CORRESPONDANCE PORT SERIE($tNumPorts;$tNomPorts)
$vNumport:=Chercher dans tableau($tNomPorts;vNomport)
 ` vNomport contient le nom du port à utiliser, il peut provenir d'une boîte de dialogue,
 ` d'une valeur stockée dans un champ, etc.
Si(tNumPorts{$vNumport}=0)
   $vNumportFinal:=0 `cas particulier sous Mac OS X
Sinon
   $vNumportFinal:=tNumPorts{$vNumport}+100
Fin de si
REGLER SERIE($vNumportFinal;params) `params contient les paramètres de communication
... `Effectuer ici les opérations souhaitées
REGLER SERIE(11) `Fermeture du port
```

RECEVOIR BUFFER

RECEVOIR BUFFER (varRéception)

Paramètre	Type		Description
varRéception	Variable texte	➡	Variable devant recevoir les données

Description

La commande **RECEVOIR BUFFER** lit les données du port série préalablement ouvert par la commande **REGLER SERIE**. Le port série comporte un buffer qui se remplit de caractères jusqu'à ce qu'une commande les charge. **RECEVOIR BUFFER** récupère les caractères présents dans le buffer, les place dans la variable *varRéception* puis vide le buffer. S'il n'y a pas de caractères dans le buffer, la variable *varRéception* est vide.

Sous Windows

Le buffer du port série sous Windows a une capacité limitée à 10 Ko. Cela signifie que le buffer peut être saturé. Lorsqu'il est plein et que de nouveaux caractères sont reçus, ils remplacent les plus anciens caractères. Les anciens caractères sont perdus ; par conséquent, il est important que le buffer soit lu rapidement lorsque des nouvelles données sont reçues.

Sous Mac OS

Le buffer du port série sous Mac OS X a une capacité en principe illimitée (elle dépend de la mémoire disponible). Si le buffer est saturé et que de nouveaux caractères sont reçus, ils remplacent les plus anciens caractères. Les anciens caractères sont perdus ; par conséquent, il est important que le buffer soit lu rapidement lorsque des nouvelles données sont reçues.

La commande **RECEVOIR BUFFER** est différente de **RECEVOIR PAQUET** dans la mesure où elle récupère tout ce qui se trouve dans le buffer et le retourne immédiatement. **RECEVOIR PAQUET**, pour sa part, attend de récupérer un caractère spécifique ou un certain nombre de caractères.

Pendant l'exécution d'un **RECEVOIR BUFFER**, l'utilisateur peut interrompre l'opération en appuyant sur les touches **Ctrl**+**Alt**+**Maj** (sous Windows) ou **Commande**+**Option**+**Maj** (sous Mac OS). Cette interruption génère une erreur -9994 que vous pouvez intercepter à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

Exemple

La méthode projet ***ECOUTER PORT SÉRIE*** utilise **RECEVOIR BUFFER** pour récupérer du texte depuis le port série et l'accumuler dans une variable interprocess :

```
   ` ECOUTER PORT SÉRIE
   ` Ouvrir le port série
REGLER SERIE(201;Vitesse 9600+Bits de données 8+Bit de stop un+Pas de parité)
◊IP_Ecouter_Port_Série:=Vrai
Tant que(◊IP_Ecouter_Port_Série)
   RECEVOIR BUFFER($vtBuffer)
   Si((Longueur($vtBuffer)+Longueur(◊vtBuffer))>MAXLARGTEXTE)
      ◊vtBuffer:=""
   Fin de si
   ◊vtBuffer:=◊vtBuffer+$Buffer
Fin tant que
```

A ce stade, tout autre process peut lire la variable interprocess *◊vtBuffer* pour exploiter les données en provenance du port série.

Pour cesser d'écouter le port série, exécutez simplement la méthode suivante :

```
   ` Fin de l'écoute du port série
◊IP_Ecouter_Port_Série:=Faux
```

Notez que l'accès à la variable interprocess *◊vtBuffer* doit être protégé par un sémaphore, de manière à ce que les process n'entrent pas en conflit (reportez-vous à la description de la fonction **Semaphore** pour plus d'informations).

RECEVOIR ENREGISTREMENT

RECEVOIR ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table dans laquelle recevoir l'enregistrement, ou Table par défaut si omis

Description

RECEVOIR ENREGISTREMENT ajoute dans *laTable* un enregistrement reçu par l'intermédiaire du port série ou d'un document ouvert par la commande **REGLER SERIE**. L'enregistrement doit avoir été envoyé par la commande **ENVOYER ENREGISTREMENT**. Lorsque vous exécutez **RECEVOIR ENREGISTREMENT**, un nouvel enregistrement est automatiquement créé dans *laTable*. Si l'enregistrement a été correctement reçu, vous pouvez le sauvegarder à l'aide de **STOCKER ENREGISTREMENT**.

L'enregistrement est reçu en totalité, ce qui signifie que les images et BLOBs stockés dans ou avec l'enregistrement sont également reçus.

Important : Lorsque des enregistrements sont envoyés et reçus par **ENVOYER ENREGISTREMENT** et **RECEVOIR ENREGISTREMENT**, la structure de la table source et celle de la table de destination doivent être compatibles. Si ce n'est pas le cas, 4D convertira les valeurs en fonction des définitions des tables lorsque **RECEVOIR ENREGISTREMENT** sera exécutée.

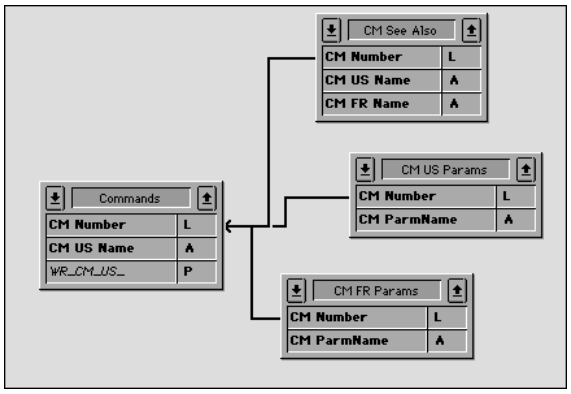
Notes :

1. Si vous recevez un enregistrement provenant d'un document avec cette commande, le document doit avoir été ouvert par la commande **REGLER SERIE**. Vous ne pouvez pas utiliser **RECEVOIR ENREGISTREMENT** avec un document ouvert par **Ouvrir document**, **Creer document** ou **Ajouter a document**.
2. Pendant l'exécution d'un **RECEVOIR ENREGISTREMENT**, l'utilisateur peut interrompre l'opération en appuyant sur les touches **Ctrl+Alt+Maj** (sous Windows) ou **Commande+Option+Maj** (sous Mac OS). Cette interruption génère une erreur -9994 que vous pouvez intercepter à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**. Généralement, vous devez gérer les interruptions d'une réception uniquement lors d'une communication série.

Exemple

L'utilisation combinée de **ENVOYER VARIABLE**, **ENVOYER ENREGISTREMENT**, **RECEVOIR VARIABLE** et **RECEVOIR ENREGISTREMENT** est idéale pour archiver des données ou échanger des données entre des bases monopostes identiques utilisées à différents endroits. Certes, vous pouvez échanger des données entre des bases 4D à l'aide des commandes d'import/export telles que **EXPORTER TEXTE** et **IMPORTER TEXTE**. Cependant, si vos données contiennent des images et/ou des tables liées, l'utilisation de **ENVOYER ENREGISTREMENT** et **RECEVOIR ENREGISTREMENT** est, de loin, plus pratique.

Par exemple, imaginons une documentation créée à l'aide de 4D et 4D Write. Comme plusieurs rédacteurs basés dans différents pays travaillent sur ce projet, nous avons besoin d'un système simple pour échanger les données entre les différentes bases. Voici une vue simplifiée de la structure de la base :



La table *[Commands]* contient la description de chaque commande ou section. Les tables *[CM US Params]* et *[CM FR Params]* contiennent respectivement les paramètres de chaque commande en anglais et en français. La table *[CM See Also]* contient les commandes indiquées en tant que Références pour chaque commande ou section. L'échange de la documentation entre les bases consiste donc à envoyer les enregistrements de *[Commands]* ainsi que leurs enregistrements liés. Pour cela, nous utilisons **ENVOYER ENREGISTREMENT** et **RECEVOIR ENREGISTREMENT**. De plus, nous utilisons **ENVOYER VARIABLE** et **ENVOYER ENREGISTREMENT** pour "cocher" les enregistrements importés/exportés.

Voici la méthode projet (simplifiée) d'export de la documentation :

```
   ` Méthode projet CM_EXPORT_SEL
   ` Cette méthode fonctionne avec la sélection courante de la table [Commands]

REGLER SERIE(12;"") ` Laissons l'utilisateur créer et ouvrir un document série
Si(OK=1)
   ` Marquons le document avec une variable décrivant son contenu
   ` Note: la variable process BUILD_LANG indique si des données US (anglaises)
   ` ou FR (françaises) sont envoyées
   $vsTag:="4DV6COMMAND"+BUILD_LANG
   ENVOYER VARIABLE($vsTag)
   ` Envoyer une variable indiquant combien de [Commands] sont exportées
   $vlNbCmd:=Enregistrements trouves([Commands])
   ENVOYER VARIABLE($vlNbCmd)
   DEBUT SELECTION([Commands])
   ` Pour chaque commande
   Boucle($vlCmd;1;$vlNbCmd)
   ` Envoyer l'enregistrement [Commands]
```

```
      ENVOYER ENREGISTREMENT([Commands])
   ` Sélection de tous les enregistrements liés
      LIEN RETOUR([Commands])
   ` En fonction de la langue, envoyer une variable indiquant
   ` le nombre de paramètres qui va suivre
      Au cas ou
         :(BUILD_LANG="US")
            $vlNbParm:=Enregistrements trouves([CM US Params])
         :(BUILD_LANG="FR")
            $vlNbParm:=Enregistrements trouves([CM FR Params])
      Fin de cas
      ENVOYER VARIABLE($vlNbParm)
   ` Envoyer les enregistrements des paramètres (s'il y en a)
      Boucle($vlParm;1;$vlNbParm)
         Au cas ou
            :(BUILD_LANG="US")
               ENVOYER ENREGISTREMENT([CM US Params])
               ENREGISTREMENT SUIVANT([CM US Params])
            :(BUILD_LANG="FR")
               ENVOYER ENREGISTREMENT([CM FR Params])
               ENREGISTREMENT SUIVANT([CM FR Params])
         Fin de cas
      Fin de boucle
   ` Envoyer une variable indiquant combien de "Références" vont suivre
      $vlNbSee:=Enregistrements trouves([CM See Also])
      ENVOYER VARIABLE($vlNbSee)
   ` Envoyer les enregistrements [See Also] (s'il y en a)
      Boucle($vlSee;1;$vlNbSee)
         ENVOYER ENREGISTREMENT([CM See Also])
         ENREGISTREMENT SUIVANT([CM See Also])
      Fin de boucle
   ` Aller à l'enregistrement [Commands] suivant et continuer l'export
      ENREGISTREMENT SUIVANT([Commands])
   Fin de boucle
   REGLER SERIE(11) ` Fermer le document
Fin de si
```

Voici la méthode projet (simplifiée) d'import de la documentation :

```
   ` Méthode projet CM_IMPORT_SEL

REGLER SERIE(10;"") ` Laissons l'utilisateur ouvrir un document existant
Si(OK=1) ` Si un document a été ouvert
   RECEVOIR VARIABLE($vsTag) ` Essayons de recevoir la variable marqueur attendue
   Si($vsTag="4DV6COMMAND@") ` Avons-nous le bon marqueur ?
      $CurLang:=Sous chaine($vsTag;Longueur($vsTag)-1) ` Extrayons la langue du marqueur
      Si(($CurLang="US") | ($CurLang="FR")) ` Avons-nous reçu un langage valide ?
         RECEVOIR VARIABLE($vlNbCmd) ` Combien de commandes dans ce document?
         Si($vlNbCmd>0) ` S'il en existe une au moins
            Boucle($vlCmd;1;$vlNbCmd) ` Pour chaque enregistrement [Commands] archivé
   ` Réception de l'enregistrement
               RECEVOIR ENREGISTREMENT([Commands])
   ` Appelons une sous-routine qui sauvegarde le nouvel enregistrement ou le copie
   ` dans un enregistrement existant
               CM_IMP_CMD($CurLang)
   ` Réception du nombre de paramètres (s'il y en a)
               RECEVOIR VARIABLE($vlNbParm)
               Si($vlNbParm>=0)
   ` Appelons une sous-routine qui appelle RECEVOIR ENREGISTREMENT puis stocke
   ` les nouveaux enregistrements ou les copie dans des enregistrements existants
                  CM_IMP_PARM($vlNbParm;$CurLang)
               Fin de si
   ` Réception du nombre de "Références" (s'il y en a)
               RECEVOIR VARIABLE($vlNbSee)
               Si($vlNbSee>0)
   ` Appelons une sous-routine qui appelle RECEVOIR ENREGISTREMENT puis stocke
   ` les nouveaux enregistrements ou les copie dans des enregistrements existants
                  CM_IMP_SEEA($vlNbSee;$CurLang)
               Fin de si
            Fin de boucle
         Sinon
            ALERTE("Le nombre de commandes dans ce document d'export est invalide.")
         Fin de si
      Sinon
         ALERTE("Le langage de ce document d'export est inconnu.")
      Fin de si
   Sinon
      ALERTE("Ce document n'est pas un document d'export.")
   Fin de si
   REGLER SERIE(11) ` Fermer document
```

```
    Fin de si
```

Notez que nous n'avons pas testé la variable OK pendant la réception des données, ni intercepté les éventuelles erreurs. Cependant, comme nous avons stocké dans le document des variables décrivant le document lui-même, si ces variables, une fois reçues, sont correctes, la probabilité d'erreur est très faible. Si par exemple un utilisateur ouvre un mauvais document, le premier test stoppe l'opération entière.

Variables et ensembles système

La variable système OK prend la valeur 1 si l'enregistrement est correctement reçu, sinon elle prend la valeur 0.

RECEVOIR PAQUET

RECEVOIR PAQUET ({docRef ;} réceptVar ; stopCar | nbOctets)

Paramètre	Type		Description
docRef	RefDoc	⇒	Numéro de référence de document ou canal courant (port série ou document)
réceptVar	Variable texte, Variable BLOB	⇒	Variable devant recevoir les données
stopCar | nbOctets	Chaîne, Entier long	⇒	Caractère(s) au(x)quel(s) stopper la réception des données ou Nombre d'octets à recevoir

Description

La commande **RECEVOIR PAQUET** lit des caractères depuis un port série ou un document.

Si *docRef* est spécifié, la commande récupère des données depuis un document ouvert par la fonction **Ouvrir document**, **Creer document** ou **Ajouter a document**. Si *docRef* est omis, la commande récupère des données depuis un port série ou un document ouvert par la commande **REGLER SERIE**.

Dans tous les cas, les caractères lus sont retournés dans la variable *réceptVar*, qui doit être une variable de type Texte, Alpha ou BLOB. Si les données ont été envoyées par la commande **ENVOYER PAQUET**, le type doit correspondre à celui du paquet envoyé.

Notes

- Si le paquet reçu est de type BLOB, la commande ne tient pas compte du jeu de caractères éventuellement défini par la commande **UTILISER FILTRE**. Le BLOB est retourné sans aucune modification.
- Si le paquet reçu est de type texte, la commande **RECEVOIR PAQUET** prend en charge les BOM (Byte Order Mark). Dans ce cas, si le jeu de caractères courant est de type Unicode (UTF-8, UTF-16 ou UTF-32), 4D tente d'identifier une BOM parmi les premiers octets reçus. Si elle est détectée, elle est filtrée de la variable *réceptVar* et 4D utilise le jeu de caractères qu'elle définit au lieu du jeu de caractères courant.

Si vous voulez recevoir un nombre prédéfini d'octets, passez ce nombre dans le paramètre *nbOctets*. Si la variable *réceptVar* est de type Texte, vous pouvez lire en un seul appel jusqu'à 2 Go de texte (limite théorique).

Si vous voulez recevoir des données jusqu'à ce qu'une chaîne de caractères (comportant un ou plusieurs caractères) soit lue, passez-la dans le paramètre *stopCar* (la chaîne n'est pas retournée dans *réceptVar*).
Dans ce cas, si la chaîne de caractères spécifiée par *stopCar* n'est pas trouvée :

- lorsque **RECEVOIR PAQUET** lit un document, l'exécution de la commande se terminera à la fin du document.
- lorsque **RECEVOIR PAQUET** lit des données en provenance du port série, la commande s'exécutera indéfiniment jusqu'à ce que le délai d'attente (s'il est fixé) soit écoulé (cf. la commande **FIXER TIMEOUT**) ou que l'utilisateur interrompe la réception (voir ci-dessous).

Pendant l'exécution d'un **RECEVOIR PAQUET**, l'utilisateur peut interrompre l'opération en appuyant sur les touches **Ctrl**+**Alt**+**Maj** (sous Windows) ou **Commande**+**Option**+**Maj** (sous Mac OS). Cette interruption génère une erreur -9994 que vous pouvez intercepter à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**. Généralement, vous devez gérer les interruptions d'une réception uniquement lors d'une communication série.

Lors de la lecture d'un document, le premier **RECEVOIR PAQUET** commence par lire le début du document. La lecture des paquets suivants débute au caractère situé immédiatement après le dernier octet lu.

Note : Ce fonctionnement est valide avec un document ouvert par **REGLER SERIE**. Cependant, pour un document ouvert par **Ouvrir document**, **Creer document** ou **Ajouter a document**, vous pouvez aussi utiliser les commandes **Position dans document** et **CHANGER POSITION DANS DOCUMENT** pour connaître et modifier la position à laquelle, dans le document, la prochaine écriture (**ENVOYER PAQUET**) ou lecture (**RECEVOIR PAQUET**) aura lieu.

En cas de tentative de lecture après la fin d'un document, **RECEVOIR PAQUET** retourne les données lues jusqu'à ce point et la variable système OK prend la valeur 1. Les **RECEVOIR PAQUET** suivants retourneront une chaîne vide et OK prendra la valeur zéro.

Exemple 1

L'exemple suivant lit 20 caractères depuis un port série et les place dans la variable **RécupVingt** :

```
RECEVOIR PAQUET(RécupVingt;20)
```

Exemple 2

L'exemple suivant lit des données depuis le document référencé par la variable **MonDoc** et les place dans la variable *vData*. La commande récupère les données jusqu'à ce qu'elle rencontre un retour chariot :

```
RECEVOIR PAQUET(MonDoc;vData;Caractere(Retour chariot))
```

Exemple 3

L'exemple suivant lit des données du document référencé par la variable **MonDoc** et les place dans la variable *vData*. La commande récupère les données jusqu'à ce qu'elle rencontre une balise HTML de fin de tableau (*</TD>*) :

```
RECEVOIR PAQUET(MonDoc;vData;"</TD>")
```

Exemple 4

L'exemple suivant lit des données d'un document et les place dans des champs. Les données sont stockées dans des champs de longueur fixe. La méthode fait appel à une sous-routine pour éliminer les espaces superflus (situés derrière les valeurs). Le code de la sous-routine est présenté après la méthode :

```
$Doc :=Ouvrir document("";"TEXT") ` Ouverture d'un document de type Texte
```

```
Si(OK=1) ` Si le document est ouvert…
   Repeter ` Boucle jusqu'à ce qu'il n'y ait plus de données
      RECEVOIR PAQUET($Doc;$Var1;15) ` Lecture de 15 caractères
      RECEVOIR PAQUET($Doc;$Var2;15) ` Même chose pour le second champ
      Si(OK=1) ` Si ce n'est pas la fin du document…
         CREER ENREGISTREMENT([Personnes]) ` Créer un nouvel enregistrement
         [Personnes]Prénom:=Elimine($Var1) ` Sauvegarder le prénom
         [Personnes]Nom:=Elimine($Var2) ` Sauvegarder le nom
         STOCKER ENREGISTREMENT([Personnes]) ` Sauvegarder l'enregistrement
      Fin de si
   Jusque(OK=0)
   FERMER DOCUMENT($Doc) ` Fermeture du document
Fin de si
```

Les espaces superflus derrière les valeurs sont éliminés par la méthode suivante, appelée **Elimine** :

```
Boucle($i;Longueur($1);1;-1) ` Boucle sur la fin de la chaîne d'où démarrer
   Si($1≤$i≥#" ") ` Si ce n'est pas un espace…
      $i :=-$i  ` Forcer la boucle à stopper
   Fin de si
Fin de boucle
$0:=Supprimer chaine($1;-$i;Longueur($1)) ` Suppression des espaces
```

Variables et ensembles système

Après un appel à **RECEVOIR PAQUET**, la variable système OK prend la valeur 1 si le paquet est reçu sans erreur. Sinon, OK prend la valeur 0.

RECEVOIR VARIABLE

RECEVOIR VARIABLE (variable)

Paramètre	Type		Description
variable	Variable	⇨	Variable dans laquelle recevoir une variable

Description

La commande **RECEVOIR VARIABLE** reçoit *variable*, une variable envoyée par la commande **ENVOYER VARIABLE**, depuis un document ou un port série préalablement ouvert par la commande **REGLER SERIE**.

En mode interprété, si la variable n'existe pas préalablement à l'appel de **RECEVOIR VARIABLE**, elle sera créée, typée et remplie en fonction de ce qui a été reçu. En mode compilé, la variable doit être du même type que celle qui est reçue. Dans les deux cas, le contenu de la variable est remplacé par celui de la variable reçue.

Notes :

1. Si vous recevez une variable depuis un document avec cette commande, le document doit avoir été ouvert par la commande **REGLER SERIE**. Vous ne pouvez pas utiliser **RECEVOIR VARIABLE** avec un document ouvert par **Ouvrir document**, **Creer document** ou **Ajouter a document**.
2. Cette commande ne fonctionne pas avec les variables de type tableau. Si vous voulez envoyer et recevoir des tableaux via un document ou un port série, utilisez les **Commandes du thème BLOB**.
3. Pendant l'exécution d'un **RECEVOIR VARIABLE**, l'utilisateur peut interrompre l'opération en appuyant sur les touches **Ctrl**+**Alt**+**Maj** (sous Windows) ou **Commande**+**Option**+**Maj** (sous Mac OS). Cette interruption génère une erreur –9994 que vous pouvez intercepter à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**. Généralement, vous devez gérer les interruptions d'une réception uniquement lors d'une communication série.

Exemple

Reportez-vous à l'exemple de la commande **RECEVOIR ENREGISTREMENT**.

Variables et ensembles système

La variable système OK prend la valeur 1 si la variable est correctement reçue, sinon elle prend la valeur 0.

REGLER SERIE

REGLER SERIE (port ; param)

Paramètre	Type		Description
port	Entier long	⇒	Numéro de port série
param	Entier long	⇒	Paramètres de communication

REGLER SERIE (opération ; nomFichier)

Paramètre	Type		Description
opération	Entier long	⇒	Opération à effectuer sur document
nomFichier	Chaîne	⇒	Nom du document

Description

La commande **REGLER SERIE** permet d'ouvrir un port série ou un document. Vous ne pouvez ouvrir qu'un port série ou un document à la fois avec cette commande.

Note historique : A l'origine, **REGLER SERIE** a été la première commande 4D permettant de travailler avec les ports série et des documents sur disque. Depuis, de nouvelles commandes ont été ajoutées. Aujourd'hui, vous pouvez généralement travailler avec des documents sur disque à l'aide des commandes **Ouvrir document**, **Creer document** et **Ajouter a document**, puis lire et écrire des caractères dans les documents avec **Creer document** et **RECEVOIR PAQUET** (ces deux commandes fonctionnent aussi avec **REGLER SERIE**). Cependant, si vous souhaitez utiliser les commandes **ENVOYER VARIABLE**, **RECEVOIR VARIABLE**, **ENVOYER ENREGISTREMENT** et **RECEVOIR ENREGISTREMENT**, vous devez appeler **REGLER SERIE** pour accéder aux documents sur disque.

La description de la commande **REGLER SERIE** se compose de deux sections :

- Travailler avec les ports série
- Travailler avec des documents

Travailler avec les ports série : REGLER SERIE(port;param)

La première syntaxe de **REGLER SERIE** ouvre un port série et définit le protocole de communication ainsi que des informations supplémentaires. Les données peuvent être envoyées par les commandes **ENVOYER PAQUET**, **ENVOYER ENREGISTREMENT** ou **ENVOYER VARIABLE**, et reçues par les commandes **RECEVOIR BUFFER**, **RECEVOIR PAQUET**, **RECEVOIR VARIABLE** ou **RECEVOIR ENREGISTREMENT**.

- Le premier paramètre, *port*, définit le port et le protocole utilisés. Vous pouvez adresser jusqu'à 99 ports série (un par un).
  Le tableau suivant liste les valeurs possibles du paramètre *port* :

Valeurs port	Description
0	Port imprimante (Mac) ou COM2 (Windows) sans protocole
1	Port modem (Mac) ou COM1 (Windows) sans protocole
20	Port imprimante (Mac) ou COM2 (Windows) avec protocole logiciel tel que XON/XOFF
21	Port modem (Mac) ou COM1 (Windows) avec protocole logiciel tel que XON/XOFF
30	Port imprimante (Mac) ou COM2 (Windows) avec protocole matériel tel que RTS/CTS
31	Port modem (Mac) ou COM1 (Windows) avec protocole matériel tel que RTS/CTS
101 à 199	Communication série sans protocole*
201 à 299	Communication série avec protocole logiciel tel que XON/XOFF*
301 à 399	Communication série avec protocole matériel tel que RTS/CTS*

Important : La valeur que vous passez dans *port* doit désigner un port série "logique" reconnu par votre système d'exploitation. Par exemple, pour que vous puissiez utiliser les valeurs 101, 203 et 325, les ports série COM1, COM3 et COM25 doivent avoir été correctement configurés.

Note sur les ports série

En standard, les systèmes Mac OS et Windows reconnaissent deux ports série logiques : sous Mac OS, le port modem et le port imprimante ; sous Windows, les ports COM1 et COM2. Toutefois, des ports série supplémentaires peuvent être ajoutés, par l'intermédiaire de cartes d'extension. 4D n'adressait à l'origine que les deux ports série standard, et a intégré par la suite la gestion des ports série supplémentaires. Pour des raisons de compatibilité, les deux systèmes d'adressage ont été conservés.

- Si vous souhaitez adresser uniquement un port série standard (imprimante/COM2 ou modem/COM1), vous pouvez passer dans le paramètre *port* soit une des valeurs 0, 1, 20, 21, 30 et 31 (correspondant à l'ancien mode de fonctionnement de 4D), soit une valeur > 100 (cf. ci-dessous).
- Si vous souhaitez adresser des ports série "étendus", vous devez passer dans *port* (pour adresser le Nième port série) la valeur N+100, augmentée éventuellement de 100 ou de 200, si vous voulez utiliser respectivement un protocole logiciel ou matériel.

Exemple 1

Vous souhaitez utiliser le port imprimante/COM2 sans protocole, vous pouvez utiliser l'une des syntaxes suivantes :

```
REGLER SERIE(0;param)
```

ou

```
REGLER SERIE(102;param)
```

Exemple 2

Vous souhaitez utiliser le port modem/COM1 avec le protocole XON/XOFF, vous pouvez utiliser l'une des syntaxes suivantes :

```
REGLER SERIE(21;param)
```

ou

```
REGLER SERIE(201;param)
```

Exemple 3

Vous souhaitez utiliser le port COM25 avec le protocole RTS/CTS, vous devez utiliser la syntaxe suivante :

```
REGLER SERIE(325;param)
```

- Le second paramètre, *param*, permet de fixer la vitesse, le nombre de bits de données, le nombre de bits de stop et la parité. La valeur de *param* se calcule en additionnant les valeurs de vitesse, de bits de données, de bits de stop et de parité, telles que définies dans le tableau ci-dessous. Par exemple, pour paramétrer la communication à 1200 bauds, 8 bits de données, 1 bit de stop et aucune parité, passez 19550 (soit 94 + 3072 + 16384 + 0) dans le paramètre *param*.

Contrôle	Valeur *param* (à cumuler)	Fonction
Vitesse (en bauds)	380	300
	189	600
	94	1200
	62	1800
	46	2400
	30	3600
	22	4800
	14	7200
	10	9600
	4	19200
	2	28800
	1	38400
	0	57600
	1022	115200
	1021	230400
Bits de données	0	5
	2048	6
	1024	7
	3072	8
Bits de stop	16384	1
	–32768	1,5
	–16384	2
Parité	0	Aucune
	4096	Impaire
	12288	Paire

Astuce : Les différentes valeurs numérique à cumuler et à passer dans les paramètres *port* et *param* (à l'exception des valeurs de COM1...COM99) sont disponibles en tant que **Constantes** prédéfinies dans le thème **Communications** de l'Explorateur, en mode Développement. Pour les valeurs de COM1...COM99, vous devez utiliser des valeurs numériques littérales.

Lorsque vous n'avez plus besoin d'un port série, vous devez le refermer. Pour cela, appelez de nouveau **REGLER SERIE** et passez-lui la valeur 11. Exemple :

```
REGLER SERIE(11) `Referme un port série préalablement ouvert
```

Travailler avec des documents : REGLER SERIE(opération;document)

La seconde syntaxe de la commande **REGLER SERIE** vous permet de créer, ouvrir ou fermer un document. A la différence des commandes du thème **Documents système**, **REGLER SERIE** ne permet d'ouvrir qu'un document à la fois. Le document peut être "lu à partir de" ou "écrit dans". Reportez-vous à la section **Présentation des documents système** pour plus d'informations sur ce point.

Le premier paramètre, *opération*, définit l'opération à effectuer avec le document désigné par *document*. Le tableau suivant dresse la liste les valeurs d'*opération* et le résultat obtenu, en fonction de la valeur de *document*.
La première colonne fournit les valeurs possibles du paramètre *opération*. La deuxième colonne fournit les valeurs possibles du paramètre *document*. La troisième colonne décrit le résultat obtenu. Par exemple, pour afficher un fichier de type texte dans une boîte de dialogue standard d'ouverture de document, vous pouvez écrire l'instruction suivante :

```
REGLER SERIE(13;"")
```

Opération	Document	Résultat
10	Chaîne	Ouvre le document dont le nom est spécifié par Chaîne. Si le document n'existe pas, il est créé et ouvert.
10	"" (chaîne vide)	Affiche la boîte de dialogue d'ouverture de fichier. Tous les types de fichiers sont présentés.
11	Aucun	Referme un fichier ouvert.
12	"" (chaîne vide)	Affiche la boîte de dialogue standard d'enregistement de fichier, permettant de créer un nouveau fichier.
13	"" (chaîne vide)	Affiche la boîte de dialogue d'ouverture de fichier. Seuls les fichiers de type Texte sont présentés.

Toutes les opérations décrites dans ce tableau modifient la variable système Document en conséquence. De plus, la variable système OK prend la valeur 1 si l'opération s'est déroulée correctement, 0 sinon.

Exemple 4

Reportez-vous aux exemples des commandes **RECEVOIR BUFFER**, **FIXER TIMEOUT** et **RECEVOIR ENREGISTREMENT**.

UTILISER FILTRE

UTILISER FILTRE (filtre {; typeFiltre})

Paramètre	Type		Description
filtre	Chaîne, Opérateur	⇒	Nom du jeu de caractères à utiliser ou * pour restaurer le jeu par défaut
typeFiltre	Entier long	⇒	0 = Filtre d'exportation, 1 = Filtre d'importation

Description

La commande **UTILISER FILTRE** permet de modifier le jeu de caractères utilisé par 4D pour toutes les opérations de transfert entre la base et un document ou un port série pour le process courant. Cela inclut les données transférées par les commandes d'import/export Texte, SYLK et DIF, ainsi que celles envoyées par les commandes **ENVOYER PAQUET** et **RECEVOIR PAQUET** (paquets de type texte) et **RECEVOIR BUFFER**. Les filtres n'ont pas d'effet sur les données transférées par les commandes **ENVOYER ENREGISTREMENT**, **ENVOYER VARIABLE**, **RECEVOIR ENREGISTREMENT**, **ENVOYER PAQUET**, et **RECEVOIR PAQUET** (paquets de type BLOB) et **RECEVOIR VARIABLE**.

Le paramètre *filtre* doit correspondre au nom “IANA” du jeu de caractères à utiliser, ou l’un de ses alias. Par exemple, les noms “iso-8859-1” ou “utf-8” sont des noms valides, ainsi que les alias “latin1” ou “l1”. Pour plus d’informations sur ces noms, reportez-vous à l’adresse http://www.iana.org/assignments/character-sets. Des exemples de noms IANA sont également fournis dans la description de la commande **CONVERTIR DEPUIS TEXTE**.

Si *typeFiltre* est égal à 0, le filtre est défini pour l'exportation. Si *typeFiltre* est égal à 1, il est défini pour l'importation. Si vous ne passez pas le paramètre *typeFiltre*, le filtre d'exportation est utilisé par défaut.

Lorsque le paramètre * est passé, le jeu de caractères par défaut est rétabli (filtre d'importation ou d'exportation, en fonction de la valeur de *typeFiltre*. Dans 4D, le jeu de caractères par défaut est UTF-8.

Exemple

L'exemple suivant (mode Unicode) utilise le jeu de caractères UTF-16 pour exporter un texte, puis le jeu de caractères par défaut est rétabli :

```
UTILISER FILTRE("UTF-16LE";0) ` Utiliser le jeu de caractères UTF-16 'Little Endian'
EXPORTER TEXTE([Ma Table];"Mon Texte") ` Exporter les données avec le filtre
UTILISER FILTRE(*;0) ` Rétablir le jeu par défaut
```

Variables et ensembles système

La variable système OK prend la valeur 1 si le filtre est correctement chargé, sinon elle prend la valeur 0.

Compilateur

Le compilateur intégré de 4D vous permet de traduire vos applications de base de données en instructions de niveau assembleur. Les avantages procurés par le compilateur sont les suivants :

- **Vitesse :** votre base de données s'exécute de 3 à 1000 fois plus vite.
- **Vérification du code :** la cohérence interne du code de votre application de base de données est entièrement contrôlée. Les conflits de logique et de syntaxe sont détectés.
- **Protection :** une fois votre base compilée, vous pouvez en supprimer le code interprété. Alors, la base compilée dispose des mêmes fonctionnalités que la base originale, à la différence près que la structure et les méthodes ne peuvent plus être visualisées ni modifiées délibérément ou par inadvertance.
- **Application indépendantes "double-cliquables" :** une base compilée peut également être transformée en application indépendante (sous Windows, des fichiers ".EXE") comportant sa propre icône.
- **Exécution en mode préemptif** : seul le code compilé peut être exécuté dans un process préemptif.

Pour une description du fonctionnement du compilateur de 4D, reportez-vous au manuel Mode Développement.

Les commandes de ce thème sont liées à l'utilisation du compilateur. Elles vous permettent de normaliser les types de données exploitées dans votre base. La commande **APPELER 4D** est utilisée spécifiquement dans les bases compilées.

C_BLOB	C_REEL	C_TEXTE
C_BOOLEEN	C_ENTIER LONG	C_DATE
C_POINTEUR	C_IMAGE	C_HEURE
C_OBJET	APPELER 4D	

Note de compatibilité : Les commandes obsolètes **_o_C_GRAPHE**, **_o_C_ENTIER** et **_o_C_ALPHA** ne doivent plus être utilisées.

A l'exception d'**APPELER 4D**, ces commandes déclarent des variables et leur assignent un type. La déclaration des variables permet de lever toute ambiguïté en ce qui concerne leur type. Lorsqu'une variable n'est pas déclarée par l'une de ces commandes, le compilateur déduit son type. Mais il lui est souvent difficile de déduire le type d'une variable utilisée dans les formulaires. Par conséquent, il est particulièrement important d'utiliser ces commandes pour déclarer les variables placées dans les formulaires.

Note : Pour gagner du temps, vous pouvez utiliser l'option de génération et de mise à jour des méthodes de typage (appelées “Méthodes compilateur”), proposée dans la fenêtre du compilateur. Cette option crée automatiquement des méthodes de typage recensant et donnant un type à l'ensemble des variables utilisées dans la base.

Les **tableaux** sont des variables devant respecter les mêmes règles que les variables standard en vue de la compilation. Les commandes de déclaration des tableaux sont groupées dans le thème **Tableaux**.

Principes généraux d'écriture de code destiné à être compilé

- Vous ne devez pas donner le même nom à des méthodes ou des variables différentes. Vous ne devez pas avoir une méthode qui aurait le même nom qu’une variable.
- Vous ne pouvez pas modifier le type d'une variable ou d'un tableau.
- Vous ne pouvez pas convertir un tableau simple en tableau à deux dimensions, et vice-versa.
- Vous ne pouvez pas modifier la longueur d'une variable chaîne ni celle des éléments d'un tableau alphanumérique.
- Bien que le compilateur déduise le type des variables si nécessaire, il est conseillé de déclarer le type des variables à l'aide des directives de compilation lorsque le type de données est ambigu, en particulier dans un formulaire.
- Une autre raison de déclarer explicitement le type des variables est l'optimisation de votre code. Cela est particulièrement vrai pour les variables utilisées comme compteurs. Dans ce cas, l'utilisation de variables de type Entier long assure un maximum de performances.
- Pour effacer une variable (c'est-à-dire l'initialiser à une valeur nulle), utilisez la commande **EFFACER VARIABLE** avec le nom de la variable. N'utilisez pas de chaîne alphanumérique pour désigner le nom de la variable avec la commande **EFFACER VARIABLE**.
- La fonction **Indefinie** retournera toujours **Faux**. Les variables sont toujours définies.
- Les opérations numériques effectuées sur des variables de type Entier long ou Entier sont généralement beaucoup plus rapides que celles effectuées sur des valeurs Numérique (réel).
- Les indirections de variables, utilisées dans la version 3 de 4D, ne sont pas permises. Vous ne pouvez pas utiliser l'indirection alphanumérique, à l'aide du symbole 'paragraphe' (§), pour référencer des variables indirectement. Vous ne pouvez pas non plus utiliser les indirections numériques, à l'aide des accolades ({...}). Les accolades ne peuvent être utilisées que pour accéder à un élément de tableau ayant été déclaré. En revanche, vous pouvez utiliser le passage de paramètres.
- Si vous avez coché la propriété "Peut être exécutée dans un process préemptif" pour la méthode, le code ne doit pas appeler de commandes thread-unsafe ou d'autres méthodes thread-unsafe.

Ces principes sont détaillés dans les sections suivantes :

- **Utilisation des directives de compilation**, expliquant quand et où écrire des directives de compilation
- **Guide du typage**, décrivant les différents types de conflits pouvant se produire lors de la compilation des bases 4D,
- **Précisions de syntaxe**, fournissant des informations supplémentaires concernant plusieurs commandes 4D,
- **Conseils d’optimisation**, proposant des conseils permettant d'accélérer l'exécution des applications en mode compilé,
- **Ecrire une méthode thread-safe**.

Exemple 1

Voici quelques déclarations de variables standard pour le compilateur :

```
C_BLOB(vxMonBlob) // La variable process vxMonBlob est déclarée avec le type BLOB
C_BOOLEEN(<>SousWindows) // La variable interprocess <>SousWindows est déclarée avec le type booléen
C_DATE($vdCurDate) // La variable locale $vdCurDate est déclarée avec le type Date
C_ENTIER LONG(vg1;vg2;vg3) // Les 3 variables process vg1, vg2 et vg3 sont déclarées avec le type entier long
```

Exemple 2

Dans cet exemple, la méthode projet *uneMéthodeParmiD'Autres* déclare 3 paramètres:

```
 // Méthode projet uneMéthodeParmiD'Autres
 // uneMéthodeParmiD'Autres ( Numérique ; Date { ; Entier long } )
 // uneMéthodeParmiD'Autres ( Montant ; Date { ; Ratio } )

C_REEL($1) // le 1er paramètre est du type Réel (Numérique)
C_DATE($2) // le 2e paramètre est du type Date
C_ENTIER LONG($3) // le 3e paramètre est du type Entier long

 // ...
```

Exemple 3

Dans l'exemple suivant, la méthode projet *ajoutCapitale* accepte un paramètre de type texte et retourne un texte :

```
 // Méthode projet ajoutCapitale
 // ajoutCapitale ( Texte ) -> Texte
 // ajoutCapitale ( Chaîne source ) -> Chaîne avec la première lettre capitale

C_TEXTE($0;$1)
$0:=Majusc(Sous chaine($1;1;1))+Minusc(Sous chaine($1;2))
```

Exemple 4

Dans l'exemple suivant, la méthode projet *envoyerPaquets* accepte un paramètre de type Heure suivi d'un nombre variable de paramètres de type Texte :

```
 ` Méthode projet envoyerPaquets
 ` envoyerPaquets ( Heure ; Texte { ; Texte2... ; TextN } )
 ` envoyerPaquets ( docRef ; Données { ; Données2... ; DonnéesN } )

C_HEURE($1)
C_TEXTE(${2})
C_ENTIER LONG($vlPaquet)

Boucle($vlPaquet;2;Nombre de paramètres)
   ENVOYER PAQUET($1;${$vlPaquet})
Fin de boucle
```

Exemple 5

Dans l'exemple suivant, la méthode projet *compiler_Param_Prédéclare28* pré-déclare la syntaxe d'autres méthodes projet, à l'intention du compilateur :

```
 ` Méthode projet compiler_Param_Prédéclare28

C_REEL(uneMéthodeParmiDautres;$1) ` uneMéthodeParmiDautres ( Réel ; Entier { ; Entier long } )
C_DATE(uneMéthodeParmiDautres;$2) ` ...
C_ENTIER LONG(uneMéthodeParmiDautres;$3) ` ...
C_TEXTE(ajoutCapitale;$0;$1) ` ajoutCapitale ( Texte ) -> Texte
C_HEURE(envoyerPaquets;$1) ` envoyerPaquets ( Heure ; Texte { ; Texte2... ; TexteN } )
C_TEXTE(envoyerPaquets;${2}) ` ...
```

Typage des variables

4D utilise trois catégories de variables :

- les variables locales,
- les variables process,
- les variables interprocess.

Pour plus d'informations sur ce point, reportez-vous à la section **Variables**. Les variables process et les variables interprocess sont structurellement de même nature pour le compilateur.

Types des variables

Toutes les variables ont un type. Comme décrit dans la section **Types de données**, vous disposez de différents types pour les variables simples :

Booléen
Date
Entier long
Graphe
Heure
Image
Numérique (ou Réel)
Pointeur
Texte
BLOB
Objet

Pour les variables de type Tableau, vous disposez des types suivants :

Tableau Booléen
Tableau Date
Tableau Entier
Tableau Entier long
Tableau Image
Tableau Numérique (ou Réel)
Tableau Heure
Tableau Objet
Tableau Pointeur
Tableau BLOB
Tableau Texte

Notes :
- Le type Objet est disponible depuis 4D v14.
- Les anciens types Alpha (chaîne de longueur fixe) et Entier ne sont plus utilisés pour les variables. Dans le code existant, ils sont automatiquement redirigés vers les types Texte et Entier long.

Création de la table des symboles

Lorsque vous travaillez avec 4D en mode interprété, une variable peut avoir plusieurs types. Cette tolérance se justifie parfaitement puisque vous êtes en mode interprété. En effet, à chaque ligne de code, 4D interprète l'instruction et comprend le contexte.
Lorsque vous travaillez en mode compilé, vous êtes dans une situation différente. Alors que l'interprétation agit ligne par ligne, la compilation s'intéresse à une base dans sa globalité.

La manière d'opérer du compilateur est la suivante :

- Le programme explore systématiquement les objets qui lui sont proposés par 4D.
  Ces objets sont les méthodes base, les méthodes projet, les méthodes formulaire, les méthodes table (triggers) et les méthodes objet.
- Le programme peigne ces objets pour retrouver le type de chacune des variables utilisées dans la base et génère la table des variables et des méthodes.
- Une fois qu'il a retrouvé le type de toutes les variables, le compilateur traduit la base. Encore faut-il qu'il puisse identifier un type d'une manière univoque pour chacune des variables.

Si le compilateur trouve un même nom de variable avec deux types différents, il n'a aucune raison de privilégier l'un par rapport à l'autre. En d'autres termes, avant de pouvoir ranger un objet, de lui donner une adresse mémoire, le compilateur a besoin de connaître l'identité exacte de cet objet, c'est-à-dire son nom et son type, le type permettant au compilateur de déduire sa taille. Ainsi, pour chaque application compilée, le compilateur crée un plan, contenant, pour chaque variable, son nom (ou identificateur), son emplacement (ou adresse mémoire) et la taille qu'elle occupe (représentée par son type). Ce "plan" s'appelle la table de symboles. Une option des Préférences vous permet de générer ou non cette table sous forme de fichier lors de la compilation.
Ce plan est également utilisé pour la génération automatique des méthodes compilateur.

Typage des variables par le compilateur

Si vous voulez que le compilateur vérifie votre typage ou bien s'en charge lui-même, placer une directive de compilation est simple. Vous avez le choix entre deux possibilités, qui correspondent d'ailleurs à deux méthodes de travail :

- ou bien vous écrivez cette directive lors de la première utilisation de la variable, qu'il s'agisse d'une variable locale, process ou interprocess. La seule recommandation en la matière est que vous l'inscriviez bien à la première utilisation de la variable dans la première méthode exécutée. Attention, lors de la compilation le compilateur prend les méthodes dans l'ordre de leur création, et non dans l'ordre dans lequel elles apparaissent dans l'Explorateur.
- ou bien, si vous êtes systématique, regroupez toutes vos variables process ou interprocess avec les directives afférentes dans la **Méthode base Sur ouverture** ou une méthode appelée par la **Méthode base Sur ouverture**.
  Pour les variables locales, regroupez ces directives en tête de la méthode où elles sont utilisées.

Valeurs par défaut

Au moment de leur typage via une directive de compilation, les variables reçoivent une valeur par défaut, qu'elles conserveront au cours de la session tant qu'elles n'auront pas été affectées.

La valeur par défaut dépend du type et de la catégorie de la variable, du contexte d'exécution (interprété ou compilé), ainsi que, pour le mode compilé, des options de compilation définies dans la **Page Compilateur** des Propriétés de la base :

- Les variables process et interprocess sont toujours positionnées "à zéro" (qui signifie selon les cas 0, chaîne vide, blob vide, pointeur nil, date 00-00-00…)
- Les variables locales sont positionnées :
  - en mode interprété : à zéro
  - en mode compilé, dépendant de l'option **Initialiser les variables locales** des Propriétés de la base :
    - à zéro lorsque "à zéro" est sélectionné,
    - à une valeur arbitraire fixe lorsque "à une valeur aberrante" est sélectionné (0x72677267 pour les numériques et les heures, toujours vrai pour les booléens), équivalent de "à zéro" pour les autres,
    - à une valeur aléatoire (pour les numériques) lorsque "non" est sélectionné.

Le tableau suivant illustre ces valeurs par défaut :

Type	Interprocess	Process	Local interprété	Local compilé "à zéro"	Local compilé "aberrant"	Local compilé "non"
Booléen	False	False	False	False	True	True (varie)
Date	00-00-00	00-00-00	00-00-00	00-00-00	00-00-00	00-00-00
Entier long	0	0	0	0	1919382119	909540880 (varie)
Graphe	0	0	0	0	1919382119	775946656 (varie)
Heure	00:00:00	00:00:00	00:00:00	00:00:00	533161:41:59	249345:34:24 (varie)
Image	picture size=0	picture size=0	picture size=0	picture size=0	picture size=0	picture size=0
Réel	0	0	0	0	1.250753659382e+243	1.972748538022e-217 (varie)
Pointeur	Nil=true	Nil=true	Nil=true	Nil=true	Nil=true	Nil=true
Texte	""	""	""	""	""	""
Blob	Blob size=0	Blob size=0	Blob size=0	Blob size=0	Blob size=0	Blob size=0
Objet	undefined	undefined	undefined	undefined	undefined	undefined

Quand utiliser des directives de compilation ?

Les directives de compilation sont utiles dans deux cas :

- lorsque le compilateur ne peut pas déduire seul le type d'une variable,
- lorsque vous voulez éviter que le compilateur fasse des déductions.

Par ailleurs, utiliser des directives de compilation peut vous permettre de réduire le temps de compilation.

Cas d'ambiguïté

Il arrive que le compilateur ne puisse pas déduire le type d'une variable, et cela pour plusieurs raisons. Il est impossible de recenser tous les cas de figure. Une chose est certaine, c'est qu'en cas d'impossibilité à compiler, le compilateur vous en donnera la raison précise ainsi que les moyens d'y remédier. On peut cependant distinguer trois causes majeures d'hésitation pour le compilateur : l'ambiguïté proprement dite, l'ambiguïté sur une déduction forcée, et l'impossibilité totale de déduire un type.

L'ambiguïté proprement dite

L'ambiguïté sur le nom de la variable est généré dans le cas suivant : le compilateur choisit la première variable qu'il rencontre et assigne arbitrairement à la suivante, de même nom mais de type différent, le type qu'il a précédemment attribué. Prenons un exemple simple :
dans une méthode A,

```
LaVariable:=Vrai
```

dans une méthode B,

```
LaVariable:="La lune est verte"
```

Dans le cas où la méthode A est compilée avant la méthode B, le compilateur considérera que **LaVariable:="La lune est verte"** est un changement de type d'une variable précédemment rencontrée. Il vous signalera qu'il y a retypage. Il génère une erreur qu'il vous appartient de corriger.

Ne vous inquiétez pas, le compilateur ne transformera pas votre variable **LaVariable:="La lune est verte"** en variable booléenne sans vous demander ce que vous en pensez !

L'ambiguïté sur une déduction forcée

Il peut arriver que le compilateur déduise un type sur un objet qui ne convient pas à son utilisation finale. Dans ce cas, vous devrez typer explicitement vos variables à l'aide des directives de compilation. Voici un exemple de cas d'ambiguïté lors de l'utilisation des listes de valeurs par défaut sur un objet : dans les formulaires, il est possible de spécifier une liste de valeurs par défaut pour les objets de type combo box, pop up menu, menu/liste déroulante, onglet, zone de défilement et liste déroulante à l'aide du bouton d'édition **Valeurs** (voir à ce sujet le manuel Mode Développement de 4D). Les valeurs par défaut sont automatiquement chargées dans un tableau dont le nom est le même que celui de l'objet.

Dans le cas simple où l'objet n'est pas utilisé dans une méthode, le compilateur peut déduire son type sans ambiguïté et l'objet sera typé en tableau texte par défaut. En revanche, dans le cas où vous devez initialiser la position de votre tableau pour son affichage dans le formulaire, vous pouvez écrire les instructions suivantes dans la méthode formulaire :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      MonPopUp:=2
      ...
Fin de cas
```

C'est dans ce cas que l'ambiguité apparaît : lors de l'analyse des méthodes, le compilateur déduira par défaut le type Réel pour la variable MonPopUp.
Dans ce cas très précis, vous devez explicitement déclarer le tableau dans une méthode compilateur ou dans la méthode formulaire :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TABLEAU TEXTE(MonPopUp;2)
      MonPopUp:=2
      ...
Fin de cas
```

L'impossibilité totale de déduire un type
Dans ce cas, seule une directive de compilation peut orienter le compilateur. Le compilateur peut se trouver dans cette situation lorsqu'une variable est utilisée sans être déclarée et dans un cadre qui ne donne aucune information sur son type possible.
Le phénomène se produit principalement dans quatre cas :

- lorsque vous utilisez des pointeurs,
- lorsque vous utilisez une commande à syntaxe multiple,
- lorsque vous utilisez une commande 4D avec des paramètres optionnels de types différents,
- lorsque vous utilisez une méthode appelée via un URL.

Cas des pointeurs
Un pointeur étant un outil universel qui a donc pour première caractéristique la flexibilité, il est inutile d'espérer qu'il renvoie un type d'une manière ou d'une autre, hormis le sien propre.
Supposons que vous écriviez dans une méthode la séquence suivante :

```
LaVar1:=5,2(1)
LePointeur:=->LaVar1(2)
LaVar2:=LePointeur->(3)
```

Bien que la ligne (2) définisse le type de la variable pointée par le pointeur LePointeur, LaVar2 n'est pas pour autant typée. Lors de la compilation, le compilateur peut reconnaître un pointeur, mais n'a aucun moyen de savoir sur quel type de variable il pointe. Il ne peut donc pas déduire le type de LaVar2 ; une directive de compilation du type **C_REEL(LaVar2)** est donc indispensable.

Cas des commandes à syntaxe multiple
Lorsque vous utilisez une variable associée à la fonction **Annee de**, la variable ne peut être, compte tenu de la nature même de la fonction, que de type Date. En revanche, prenons un cas extrême : la commande **LIRE PROPRIETES CHAMP** admet deux syntaxes :
LIRE PROPRIETES CHAMP(NoTable;NoChamp;Type;Longueur;Indexée)
LIRE PROPRIETES CHAMP(Pointeur_Champ;Type;Longueur;Indexée)

Lorsque vous utilisez cette commande, le compilateur ne peut pas deviner quelle syntaxe et quels paramètres vous avez choisis. Il vous appartient alors d'orienter le compilateur par une directive de compilation.

Cas des commandes 4D ayant des paramètres optionnels de types différents
Lorsque vous utilisez une commande 4D qui accepte plusieurs paramètres optionnels de différents types, le compilateur ne peut pas deviner quels paramètres ont été passés.
Par exemple, la commande **INFORMATION ELEMENT** admet deux paramètres optionnels. Le premier est de type Entier long, le second est de type Booléen.
La commande peut donc être utilisée comme ceci :
INFORMATION ELEMENT (liste;position;num;texte;sous-liste;déployé)
ou comme cela :
INFORMATION ELEMENT (liste;position;num;texte;déployé)
Vous devez donc utiliser des directives de compilation pour typer les paramètres optionnels passés à la commande (s'ils n'ont pas déjà été typés suite à leur utilisation dans un autre endroit de la base).

Cas des méthodes appelées via un URL
Si vous écrivez des méthodes appelées via un URL, il est nécessaire de déclarer explicitement la variable Texte $1 dans vos méthodes, par l'intermédiaire de l'instruction **C_TEXTE($1)**, dans le cas où vous n'utilisez pas $1 dans la méthode. En effet, le compilateur ne peut pas deviner qu'une méthode 4D va être appelée via un URL.

Réduction du temps de compilation

Si toutes les variables utilisées dans votre base sont explicitement déclarées, il n'est pas nécessaire que le compilateur refasse tout le typage. Dans ce cas, vous pouvez lui demander d'effectuer uniquement la phase de traduction de vos méthodes dans le chemin de compilation. Ainsi, vous économiserez environ 50 % du temps de compilation.

Cas d'optimisation

Les directives de compilation peuvent vous aider à accélérer vos méthodes. Pour plus de précision à ce sujet, reportez-vous à la section **Conseils d'optimisation**. Pour nous en tenir à un exemple simple dans cette section de présentation, imaginez que vous incrémentiez un compteur. Si vous n'avez pas déclaré la variable, le compilateur considérera par défaut qu'elle est de type Numérique (ou réel). Si vous prenez soin de préciser qu'il s'agit d'un Entier long, l'exécution de la base compilée sera plus satisfaisante. En effet, un Réel occupe, sur PC par exemple, 8 octets en mémoire alors que si vous choisissez un type Entier long, le compteur n'en occupera que 4. Il est bien évident que l'incrémentation d'un compteur de 8 octets est plus longue que celle d'un compteur de 4 octets.

Où placer les directives de compilation ?

Vous avez deux possibilités selon que vous voulez que le compilateur vérifie ou non votre typage.

Typage des variables

Le compilateur doit respecter les critères d'identification des variables.

Il existe deux possibilités :
1) Si les variables ne sont pas typées, le compilateur s'en occupera automatiquement pour vous. Toutes les fois que c'est possible, dans la mesure où il n'y a pas d'ambiguïtés, le compilateur déduit automatiquement pour vous le type des variables utilisées. Si, par exemple, vous écrivez :

```
V1:=Vrai
```

le compilateur pourra en déduire que la variable V1 est de type Booléen. De même, si vous écrivez :

```
V2:="Ceci est une phrase exemple"
```

le compilateur en déduira que V2 est une variable de type Texte.
Le programme peut également déduire le type des variables dans des cas moins faciles :

```
V3:=V1 `V3 est du même type que V1
V4:=2*V2 `V4 est du même type que V2
```

Le compilateur déduit également le type de vos variables d'après les appels aux commandes 4D et à vos propres méthodes. Si vous passez à une méthode un paramètre de type Booléen et un paramètre de type Date, le compilateur donnera le type Booléen et le type Date aux variables locales $1 et $2 de la méthode appelée.
Lors de ces déductions, sauf indication contraire dans les Propriétés de la base, le compilateur donne toujours le type le plus large possible à vos variables.
Si vous écrivez :

```
LeNombre:=4
```

le compilateur donnera à la variable LeNombre le type Réel, même si en toute rigueur la valeur 4 est entière. En d'autres termes, le compilateur n'exclut pas que dans d'autres occurrences de la variable, la valeur puisse être 4,5.

Si vous ne voulez pas de cette interprétation générale, vous pouvez préciser vos choix : c'est l'objet de ce qu'on appelle les directives de compilation.

2) Les directives de compilation servent à déclarer de façon explicite les variables simples que vous utilisez dans vos bases.
Leur utilisation se fait de la façon suivante :

```
C_BOOLEEN(Var)
```

Par cette directive, vous forcez le compilateur à créer une variable Var dont le type sera Booléen.
Dans le cas où une application comporte des directives de compilation, le compilateur les détecte et n'a pas à se livrer à une quelconque estimation. Une directive a la priorité sur une déduction à partir d'une affectation ou d'une utilisation.
Les variables simples déclarées par la directive de compilation **_o_C_ENTIER** sont considérés comme des Entiers longs, compris entre –2147483648 et +2147483647 comme pour les variables déclarées par la directive **C_ENTIER LONG**.

Typage assuré par vos soins

Si vous voulez que le compilateur n'ait pas à vérifier votre typage, vous devez lui donner les clés de l'identification de ses objets.
La convention à respecter est la suivante : les directives de compilation des variables process ou interprocess, ainsi que les paramètres, devront être placées dans une ou plusieurs méthodes dont le nom commence par **Compiler**.
Par défaut, le compilateur vous permet de générer automatiquement cinq types de méthodes Compilateur regroupant les directives pour les variables, les tableaux et les paramètres des méthodes (pour plus d'informations sur ce point, reportez-vous au manuel Mode Développement).

Note : La déclaration des paramètres des méthodes obéit à la syntaxe suivante : **Directive (nom de méthode; param)**. Cette syntaxe n'est pas exécutable en mode interprété.

Paramètres particuliers

- Les paramètres reçus par les méthodes base
  Ces paramètres sont typés par défaut par le compilateur, si la déclaration n'a pas été faite explicitement. Néanmoins, si vous les déclarez, la déclaration doit se faire à l'intérieur des méthodes base.
  La déclaration de ces paramètres ne peut pas se faire dans une méthode compilateur.
  Exemple : la **Méthode base Sur connexion Web** reçoit six paramètres $1 à $6 de type texte. En début de méthode, vous devez écrivez : **C_TEXTE** ($1;$2;$3;$4;$5;$6)

- Les triggers
  Le paramètre $0 (Entier long), résultat d'un trigger, est typé par défaut par le compilateur, si la déclaration n'a pas été faite explicitement. Néanmoins si vous le déclarez, la déclaration doit se faire à l'intérieur du trigger. La déclaration de ce paramètre ne peut pas se faire dans une méthode compilateur.

- Les objets acceptant l'événement formulaire "Sur glisser"
  Le paramètre $0 (Entier long), résultat d'un événement formulaire "Sur glisser", est typé par défaut par le compilateur, si la déclaration n'a pas été faite explicitement. Néanmoins si vous le déclarez, la déclaration doit se faire à l'intérieur de la méthode objet. La déclaration de ce paramètre ne peut pas se faire dans une méthode compilateur.
  Note : Le compilateur n'initialise pas le paramètre $0. Dès que vous utilisez l'événement formulaire Sur glisser, vous devez donc initialiser $0. Par exemple :

```
C_ENTIER LONG($0)
Si(Evenement formulaire=Sur glisser)
   $0:=0
   ...
   Si($TypeDeDonnées=Est une image)
      $0:=-1
   Fin de si
   ...
Fin de si
```

Une liberté permise par le compilateur

Les directives de compilation lèvent toute ambiguïté sur les types. L'exigence de rigueur ne se fait pas pour autant intolérance.
S'il vous arrive d'utiliser un réel là où vous avez déclaré un entier, le compilateur ne considère pas qu'il y a conflit et suit simplement vos directives. Ainsi, si

vous écrivez :

```
C_ENTIER LONG(vEntier)
vEntier:=2,6
```

Le compilateur ne verra pas un conflit de types de nature à empêcher la compilation et prendra automatiquement en compte la partie entière arrondie du nombre affecté (3 au lieu de 2,6).

Cette section décrit les principales causes de conflits de types sur les variables, ainsi que les manières de les éviter.

Conflits sur les variables simples

Les conflits de types simples peuvent se résumer comme suit :

- conflit entre deux utilisations,
- conflit entre une utilisation et une directive de compilation,
- conflit par retypage implicite,
- conflit entre deux directives de compilation.

Conflit entre deux utilisations

Le conflit de types le plus simple est celui pour lequel un même nom de variable désigne deux objets différents. Imaginez que dans une application, vous écriviez :

```
LaVariable:=5
```

et que, quelque part ailleurs, dans la même application, vous écriviez :

```
LaVariable:=Vrai
```

Vous générez un conflit de types. Le remède est simple : renommez l'une des deux variables.

Conflit entre une utilisation et une directive de compilation

Imaginez que dans une application, vous écriviez :

```
LaVariable:=5
```

et que, quelque part ailleurs, dans la même application, vous écriviez :

```
C_BOOLEEN(LaVariable)
```

Le compilateur, peignant d'abord les directives de compilation, fera de LaVariable un Booléen mais lorsqu'il découvrira :

```
LaVariable:=5
```

il signalera un conflit de types. Ici encore, le remède est simple : renommez votre variable ou modifiez la directive de compilation.

L'utilisation de variables de types différents dans une expression génère des incohérences. Le compilateur signale très logiquement les incompatibilités. Prenons un exemple simple. Vous écrivez :

```
vBooléen:=Vrai `Le compilateur déduit que vBooléen est de type Booléen
C_ENTIER LONG(<>vEntier) `Déclaration d'un Entier long par une directive de compilation
<>vEntier:=3 `Commande compatible avec la directive de compilation
LaVar:=<>vEntier+vBooléen `Opération contenant des variables dont les types sont incompatibles
```

Conflit par retypage implicite

Certaines fonctions renvoient des variables d'un type bien précis. L'affectation du résultat d'une de ces variables à une variable déjà typée différemment provoquera un conflit de types si vous ne faites pas attention.
Dans une application interprétée, vous pouvez écrire :

```
No_Ident:=Demander("Numéro d'identification") `No_Ident est de type Texte
Si(Ok=1)
   No_Ident:=Num(No_Ident) `No_Ident est de type Numérique
   CHERCHER([Personnes]Id=No_Ident)
Fin de si
```

Vous générez ici un conflit de type sur la troisième ligne. Le remède consiste à contrôler le comportement de la variable. Dans certains cas, vous aurez à créer des variables intermédiaires d'un nom différent. Dans d'autres cas, comme celui-ci en particulier, vous pouvez structurer différemment votre méthode :

```
No_Ident:=Num(Demander("Numéro d'identification")) `No_Ident est de type Numérique
Si(Ok=1)
   CHERCHER([Personnes]Id=No_Ident)
Fin de si
```

Conflit entre deux directives de compilation

Déclarer deux fois la même variable par deux directives de compilation différentes constitue, bien sûr, un retypage. Si, dans la même base, vous écrivez :

```
C_BOOLEEN(LaVariable)
C_TEXTE(LaVariable)
```

le compilateur est confronté à un dilemme et vous demande quelles étaient vos intentions. Le remède est simple : renommez l'une des deux variables.

Note sur les variables locales

Les conflits de types pour les variables locales sont absolument identiques aux conflits de types pour les variables process et interprocess, à ceci près que ces conflits se déroulent dans un espace plus restreint.
Les conflits se jouent au niveau général de la base pour les variables process et interprocess. Les conflits se jouent au niveau particulier de la méthode pour les variables locales. Vous ne pouvez donc pas écrire dans la même méthode :

```
$Temp:="Bonjour"
```

et puis plus loin

```
$Temp:=5
```

En revanche, vous pouvez écrire dans une méthode M1 :

```
$Temp:="Bonjour"
```

et dans une méthode M2 :

```
$Temp:=5
```

Conflits sur les variables tableau

Les conflits possibles pour un tableau ne portent jamais sur la taille du tableau. En mode compilé comme en mode interprété, les tableaux sont gérés dynamiquement. La taille d'un tableau peut varier au fil des méthodes et vous n'avez pas, bien sûr, à déclarer une taille maximale pour un tableau.
Vous pouvez, en conséquence, dimensionner un tableau à zéro, ajouter ou retirer des éléments, en effacer le contenu. Dans l'optique de la compilation, et ce, dans une même méthode, pour un tableau local ou dans toute la base pour un tableau process ou interprocess, vous devez veiller aux points suivants :

- ne pas changer le type des éléments du tableau,
- ne pas changer le nombre de dimensions d'un tableau,
- dans le cas des tableaux Alpha, ne pas changer la longueur des chaînes de caractères.

Changement de type des éléments d'un tableau

Si vous déclarez un tableau comme étant un tableau d'Entiers, il doit rester un tableau d'Entiers pour toute la base. Il ne pourra jamais contenir, par exemple, des éléments de type Booléen.
Si, dans une application, vous écrivez :

```
TABLEAU ENTIER(LeTableau;5)
TABLEAU BOOLEEN(LeTableau;5)
```

le compilateur ne peut identifier pour vous le type de LeTableau.
Renommez simplement l'un des deux tableaux.

Changement du nombre de dimensions d'un tableau

En version interprétée, il peut vous arriver de changer le nombre de dimensions d'un tableau.
Lorsque le compilateur établit sa table des symboles, il gère différemment les tableaux à une dimension et les tableaux à deux dimensions.
En conséquence, un tableau déclaré une fois comme étant un tableau à une dimension ne peut être re-déclaré ou utilisé comme un tableau à deux dimensions et vice versa.
Dans une même base, vous ne pouvez donc pas avoir :

```
TABLEAU ENTIER(LeTableau1;10)
TABLEAU ENTIER(LeTableau1;10;10)
```

En revanche, vous pouvez, évidemment, avoir dans la même application

```
TABLEAU ENTIER(LeTableau1;10)
TABLEAU ENTIER(LeTableau2;10;10)
```

Par ailleurs, vous pouvez parfaitement écrire :

```
TABLEAU BOOLEEN(LeTableau;5)
TABLEAU BOOLEEN(LeTableau;10)
```

Comme vous pouvez le noter dans nos exemples, c'est le nombre de dimensions d'un tableau qu'on ne peut changer en cours d'application et non la valeur des dimensions du tableau.

Note : Un tableau à deux dimensions est en fait un ensemble de plusieurs tableaux à une dimension. Pour plus de précisions, reportez-vous à la section **Tableaux à deux dimensions**.

Retypages implicites

Lors de l'utilisation des commandes **COPIER TABLEAU**, **LISTE VERS TABLEAU**, **TABLEAU VERS LISTE**, **SELECTION VERS TABLEAU**, **SELECTION LIMITEE VERS TABLEAU**, **TABLEAU VERS SELECTION**, ou **VALEURS DISTINCTES**, vous pouvez, volontairement ou involontairement, être conduit à des changements de type d'éléments ou de nombre de dimensions. Vous vous retrouverez donc dans un des cas cités précédemment.
Le compilateur vous délivrera un message d'erreur et la correction que vous aurez à faire sera en général évidente. Des exemples de retypage implicite de tableaux sont fournis dans la section **Précisions de syntaxe**.

Tableaux locaux

Si vous souhaitez compiler une base de données qui utilise des tableaux locaux (tableaux visibles uniquement par les méthodes qui les ont créés), il est nécessaire de les déclarer explicitement dans 4D avant de les utiliser.

La déclaration explicite d'un tableau signifie l'utilisation d'une commande de type **TABLEAU REEL**, **TABLEAU ENTIER**, etc.
Par exemple, si une méthode génère un tableau local d'entiers contenant 10 valeurs, vous devez écrire au préalable la ligne suivante :

```
TABLEAU ENTIER($MonTableau;10)
```

Typage des variables dessinées dans les formulaires

Les variables dessinées dans un formulaire, qu'il s'agisse d'une case à cocher ou d'une zone externe, sont toutes des variables soit process, soit interprocess.
En mode interprété, la question des types de ces variables ne se pose pas dans la pratique. Elle peut se poser, en revanche, dans l'optique d'une application compilée. Les règles du jeu sont cependant transparentes :

- soit vous avez typé votre variable,
- soit le compilateur lui attribue un type par défaut qui peut être défini dans les Préférences de compilation (voir le manuel Mode Développement).

Variables considérées par défaut comme des Numériques

Les variables suivantes sont considérées comme des Numériques par défaut :

Case à cocher
Case a cocher 3D
Bouton
Bouton inversé
Bouton invisible
Bouton 3D
Bouton image
Grille de boutons
Bouton radio
Bouton radio 3D
Radio image
Menu image
Menu déroulant hiérarchique
Liste hiérarchique
Règle
Cadran
Thermomètre
List box (type sélection)

Note : Les variables de type Règle, Cadran et Thermomètre seront toujours typées comme des Numériques, même si vous avez choisi l'option Entier long par défaut pour le type des boutons dans les Préférences.

Pour ces variables, vous ne pouvez jamais vous trouver dans un conflit de types puisqu'elles ne peuvent avoir d'autres types que ce type-là, qu'on soit en interprété ou en compilé.
Les seuls conflits de types possibles sur l'une de ces variables viendraient du fait que le nom d'une variable serait le même que celui d'une autre variable placée à un autre endroit dans l'application. Dans ce cas, le remède consiste à renommer cette deuxième variable.

Variable Zone externe

Une zone externe est toujours un Entier long. Il ne peut jamais y avoir de conflit de types.
Les seuls conflits de types possibles sur une variable Zone externe viendraient du fait que le nom de cette variable serait le même que celui d'une autre variable placée à un autre endroit dans l'application. Dans ce cas, le remède consiste à renommer cette deuxième variable.

Variable Zone 4D Write Pro

Les zones 4D Write Pro sont toujours des variables de type objet. Il ne peut pas y avoir de conflit de type, hormis si le même nom de variable a été utilisé dans un autre endroit de l'application.

Variables considérées par défaut comme du Texte

Ces variables sont les suivantes :

Variable non-saisissable,
Variable saisissable,
Liste déroulante,
Menu/liste déroulante,
Zone de défilement,
Combo box,
Pop-up Menu,
Onglet,
Zone Web
Colonne de list box (type tableau).

Ces variables se divisent en deux catégories :
• les variables simples (variables saisissables et variables non-saisissables),
• les variables d'affichage de tableaux (listes déroulantes, menus/listes déroulantes, zone de défilement, pop-up menu, combo box, onglets, colonnes de list box).

- Variables simples
  Par défaut, ces variables reçoivent le type Texte. Si elles sont utilisées dans une méthode objet ou une méthode formulaire, c'est le type que vous avez choisi qui leur sera attribué.
  Vous n'avez aucun risque de conflit de types autre que celui qui serait généré par une attribution préalable du même nom à une autre variable.

- Variables d'affichage de tableaux
  De nombreuses variables vous servent à afficher des tableaux dans les formulaires. Si les valeurs par défaut ont été saisies au niveau des contrôles

de saisie des variables en mode Développement, il est conseillé de typer les variables correspondantes explicitement par une déclaration de type tableau (**TABLEAU BOOLEEN**, **TABLEAU TEXTE**...).

List box

Chaque list box ajoute plusieurs variables dans les formulaires. Le type par défaut des variables dépend du type de list box :

	List box type tableau	List box type sélection
List box	Tableau booléen	Numérique (non utilisé)
Colonne de list box	Tableau texte	Typage dynamique
En-tête	Numérique	Numérique
Pied	Typage dynamique	Typage dynamique
Tableau de contrôle des lignes	Tableau booléen (Tableau Entier long accepté)	-
Styles	Tableau Entier long	Entier long
Couleurs de police	Tableau Entier long	Entier long
Couleurs de fond	Tableau Entier long	Entier long

Veillez à identifier et typer correctement ces variables et tableaux pour ne pas générer de conflit à la compilation.

Questions de type autour des pointeurs

Si vous utilisez des pointeurs dans vos applications, vous avez pu profiter de la puissance et de la flexibilité de cet outil dans 4D. Le compilateur en conserve intégralement les avantages.
Un pointeur peut pointer indifféremment sur une table, une variable ou un champ. Un même pointeur peut pointer sur des variables de type différent : veillez à ne pas générer des conflits artificiellement, en attribuant à une même variable des types différents.
Faites simplement attention à ne pas changer en cours de route le type de la variable sur laquelle pointe le pointeur en faisant, par exemple, une manipulation du type suivant :

```
LaVariable:=5,3
LePointeur:=->LaVariable
LePointeur->:=6,4
LePointeur->:=Faux
```

Dans ce cas de figure, votre pointeur dépointé est une variable Numérique. En affectant à cette variable une valeur booléenne, vous provoquez un conflit de types.
Si vous avez besoin dans une même méthode d'utiliser les pointeurs pour des propos différents, prenez soin de définir votre pointeur :

```
LaVariable:=5,3
LePointeur:=->LaVariable
LePointeur->:=6,4
LeBool:=Vrai
LePointeur:=->LeBool
LePointeur->:=Faux
```

Un pointeur n'a aucune existence propre. Il est toujours défini en fonction de l'objet qu'il représente. C'est pourquoi le compilateur ne peut pas détecter des conflits de types générés par une utilisation sans contrôle des pointeurs. Vous n'aurez donc pas, en cas de conflit, de message d'erreur durant la phase de typage ou celle de compilation.
Cela ne veut pas dire que lorsque vous utilisez des pointeurs, vous travaillez sans filet. Le compilateur peut vérifier vos utilisations de pointeurs lorsque vous cochez l'option **Contrôle d'exécution** dans les Préférences de compilation (cf. manuel Mode Développement).

Plug-ins

Généralités

Le compilateur a besoin, au moment de la compilation, des définitions des commandes des plug-ins utilisées dans la base à compiler, c'est-à-dire du nombre et du type des paramètres de ces commandes. Les risques d'erreur de typage sont inexistants à partir du moment où le compilateur trouve effectivement dans l'application ce que vous avez déclaré.

Assurez-vous que vos plug-ins sont installés dans le dossier **PlugIns**, à l'un des emplacements autorisés par 4D : à côté du fichier de structure de la base ou à côté de l'application exécutable (Windows) / dans le progiciel (Mac OS). Pour des raisons de compatibilité, il reste possible d'utiliser un dossier **Win4DX** ou **Mac4DX** à côté du fichier de structure. Pour plus d'informations, reportez-vous au Guide d'installation de 4D.
Le compilateur ne duplique pas le fichier mais tient compte des déclarations des commandes sans rien vous demander en supplément.
Si vos plug-ins sont placés ailleurs, le compilateur vous demande de les localiser lors du typage, via une boîte de dialogue d'ouverture de documents.

Routines recevant des paramètres implicites

Certains plug-ins, par exemple 4D Write, utilisent des commandes qui provoquent l'appel implicite à des commandes 4D.
Prenons un exemple avec 4D Write. Vous disposez d'une commande appelée **WR APPELER SUR EVENEMENT**. La syntaxe de cette commande est : **WR APPELER SUR EVENEMENT(LaZone;Evénement;MéthodeEvénement)**

Le dernier paramètre que vous passez à cette routine est le nom d'une méthode, que vous aurez vous-même écrite dans 4D. Cette méthode sera appelée par 4D Write chaque fois que l'événement attendu sera reçu et cette méthode recevra automatiquement les paramètres suivants :

Paramètres	Type	Description
$0	Entier long	Retour de fonction
$1	Entier long	Zone 4D Write
$2	Entier long	Touche Maj.
$3	Entier long	Touche Alt (Windows), Option (Mac OS)
$4	Entier long	Touche Ctrl (Windows), Commande (Mac OS)
$5	Entier long	Type d'événement qui a provoqué l'appel
$6	Entier long	Valeur variant en fonction du paramètre Evénement

Afin que le compilateur connaisse l'existence de ces paramètres et les prenne en compte, vous devez vous assurer qu'ils peuvent effectivement être typés, soit par une directive de compilation, soit parce que leur utilisation, suffisamment explicite, permet de déduire leur type.

Composants 4D

4D permet de créer et de manipuler des composants. Un composant 4D est un ensemble d'objets 4D représentant une ou plusieurs fonctionnalité(s) groupée(s) dans un fichier de structure (appelé base matrice), qu'il est possible d'installer dans différentes bases (appelées bases hôtes).
Une base hôte exécutée en mode interprété peut utiliser indifféremment des composants interprétés ou compilés. Il est possible d'installer des composants interprétés et compilés dans la même base hôte. En revanche, une base hôte exécutée en mode compilé ne peut pas utiliser de composant interprété. Dans ce cas, seuls des composants compilés peuvent être employés.
Une base hôte interprétée contenant des composants interprétés peut être compilée si elle ne fait pas appel à des méthodes du composant interprété. Dans le cas contraire, une boîte de dialogue d'alerte apparaît lorsque vous tentez de compiler l'application et la compilation est impossible.

Un conflit de nom peut se produire lorsqu'une méthode projet partagée du composant a le même nom qu'une méthode projet de la base hôte. Dans ce cas, lorsque du code est exécuté dans le contexte de la base hôte, c'est la méthode de la base hôte qui est appelée. Ce principe permet de "masquer" une méthode du composant avec une méthode personnalisée (par exemple pour obtenir une fonctionnalité différente). Lorsque le code est exécuté dans le contexte du composant, c'est la méthode du composant qui est appelée. Un masquage est signalé par un warning lors de la compilation de la base hôte.

Si deux composants partagent des méthodes du même nom, une erreur est générée au moment de la compilation de la base hôte.

Pour plus d'informations sur les composants, reportez-vous au manuel Mode Développement.

Manipulation des locales $0…$N et passation des paramètres

Les manipulations des variables locales suivent toutes les règles déjà énoncées. De même que toutes les autres variables, elles ne peuvent être retypées en cours de méthode.
Dans ce paragraphe, nous abordons deux cas de figure où l'inattention pourrait conduire à des conflits de types :

- Lorsque vous avez en fait besoin d'un retypage. L'utilisation de pointeurs vous permet alors d'éviter les conflits de types.
- Lorsque vous avez besoin d'adresser des paramètres par indirection.

Utiliser les pointeurs pour éviter les retypages

Il n'est pas possible de retyper une variable. Il est, en revanche, tout à fait possible de faire pointer un pointeur successivement sur des variables de type différent.
Un exemple nous permet d'illustrer ce propos : écrivons une fonction qui nous renvoie l'occupation mémoire d'un tableau à une dimension suivant son type. Le résultat est un numérique dans tous les cas sauf deux : dans le cas des tableaux Texte et des tableaux Image, la taille mémoire occupée par le tableau dépend de valeurs inexprimables sous forme numérique (cf. section **Tableaux et mémoire**).
Dans le cas des tableaux Texte et des tableaux Image, nous renverrons comme résultat une chaîne de caractères. Cette fonction requiert un paramètre : un pointeur sur le tableau dont on veut connaître l'occupation mémoire.
Pour effectuer cette opération, vous avez le choix entre deux méthodes :

- ne travailler qu'avec des variables locales et ne pas vous soucier de leur type — mais dans ce cas, la méthode ne fonctionnera qu'en mode interprété.
- utiliser des pointeurs et alors travailler indifféremment en interprété et en compilé.

Fonction OccupMém en interprété seulement (exemple pour Macintosh)

```
$Taille:=Taille tableau($1->)
$Type:=Type($1->)
Au cas ou
   :($Type=Est un tableau numérique)
      $0:=8+($Taille*10) ` $0 est un Numérique
   :($Type=Est un tableau entier)
      $0:=8+($Taille*2)
   :($Type=Est un tableau entierlong)
      $0:=8+($Taille*4)
   :($Type=Est un tableau date)
      $0:=8+($Taille*6)
   :($Type=Est un tableau texte)
      $0:=Chaine(8+($Taille*4))+("+Somme des longueurs des textes") ` $0 est un Texte
   :($Type=Est un tableau image)
      $0:=Chaine(8+($Taille*4))+("+Somme des tailles des images") ` $0 est un Texte
   :($Type=Est un tableau pointeur)
      $0:=8+($Taille*16)
   :($Type=Est un tableau booléen)
      $0:=8+($Taille/8)
Fin de cas
```

Dans la méthode ci-dessus, il y a changement de type de $0 suivant les valeurs de $1.
Fonction OccupMém en mode interprété et compilé (exemple pour Macintosh)
Ecrivons maintenant cette méthode en utilisant un pointeur :

```
$Taille:=Taille tableau($1->)
$Type:=Type($1->)
VarNum:=0
Au cas ou
   :($Type=Est un tableau numérique)
      VarNum:=8+($Taille*10) ` VarNum est un Numérique
   :($Type=Est un tableau entier)
      VarNum:=8+($Taille*2)
   :($Type=Est un tableau entierlong)
      VarNum:=8+($Taille*4)
   :($Type=Est un tableau date)
      VarNum:=8+($Taille*6)
   :($Type=Est un tableau texte)
      VarText:=Chaine(8+($Taille*4))+("+Somme des longueurs des textes")
   :($Type=Est un tableau image)
```

```
      VarText:=Chaine(8+($Taille*4))+("+Somme des tailles des images")
   :($Type=Est un tableau pointeur)
      VarNum:=8+($Taille*16)
   :($Type=Est un tableau booléen)
      VarNum:=8+($Taille/8)
Fin de cas
Si(VarNum#0)
   $0:=->VarNum
Sinon
   $0:=->VarText
Fin de si
```

Il faut noter la différence entre ces deux séquences :

- dans le premier cas, le résultat de la fonction est la variable que l'on attendait,
- dans le second cas, le résultat de la fonction est un pointeur sur cette variable. Il vous appartient alors de simplement dépointer le résultat reçu.

Indirections sur les paramètres

Le compilateur gère la puissance et la souplesse de l'indirection sur les paramètres. En mode interprété, 4D vous donne toute latitude concernant le nombre et le type des paramètres. Vous gardez en mode compilé cette même liberté à condition de ne pas introduire de conflit de types et de ne pas utiliser, dans la méthode appelée, plus de paramètres que vous en avez passés, ce qui est facile.
Afin de contourner un éventuel conflit, les paramètres adressés par indirection doivent tous être du même type.
Pour une bonne gestion de cette indirection, il est important de respecter la convention suivante : si tous les paramètres ne sont pas adressés par indirection, ce qui est le cas le plus fréquent, il faut que les paramètres adressés par indirection soient passés en fin de liste.
A l'intérieur de la méthode, l'adressage par indirection se fait sous la forme : ${$i}, $i étant une variable numérique. ${$i} est appelé paramètre générique.

Illustrons notre propos par un exemple : écrivons une fonction qui prend des valeurs, fait leur somme et renvoie cette somme formatée suivant un format qui peut varier avec les valeurs.
A chaque appel à cette méthode, le nombre de valeurs à additionner peut varier. Il faudra donc passer comme paramètre à notre méthode les valeurs, en nombre variable, et le format, exprimé sous forme d'une chaîne de caractères.
Un appel à cette fonction se fera de la façon suivante :

```
Résultat:=LaSomme("##0,00";125,2;33,5;24)
```

La méthode appelante récupérera dans ce cas la chaîne : 182,70, somme des nombres passés, formatée suivant le format spécifié. D'après ce que l'on a vu plus haut, les paramètres de la fonction doivent être passés dans un ordre précis : le format d'abord et ensuite les valeurs, dont le nombre peut varier d'un appel à l'autre.

Examinons maintenant la fonction que nous appelons LaSomme :

```
$Somme:=0
Boucle($i;2;Nombre de paramètres)
   $Somme:=$Somme+${$i}
Fin de boucle
$0:=Chaine($Somme;$1)
```

Cette fonction pourra être appelée de diverses manières :

```
Résultat:=LaSomme("##0,00";125,2;33,5;24)
Résultat:=LaSomme("000";1;18;4;23;17)
```

De même que pour les autres variables locales, la déclaration du paramètre générique par directive de compilation n'est pas obligatoire. Si elle est nécessaire (cas d'ambiguïté ou d'optimisation), elle se fait avec la syntaxe suivante :

```
C_ENTIER LONG(${4})
```

La commande ci-dessus signifie que tous les paramètres à partir du quatrième (inclus) seront adressés par indirection. Ils seront tous de type Entier long. Les types de $1, $2 et $3 pourront être quelconques. En revanche, si vous utilisez $2 par indirection, le type utilisé sera le type générique. Il sera donc de type Entier long, même si pour vous, par exemple, il était de type Réel.

Note : Le compilateur utilisant cette commande durant le typage, le nombre, dans la déclaration, doit toujours être une constante et jamais une variable.

Variables réservées et constantes

Des variables et des constantes de 4D ont un type et une identité fixés par le compilateur. On ne peut donc créer une nouvelle variable, une méthode, une fonction ou une commande de plug-in portant le nom d'une de ces variables ou d'une de ces constantes. Bien entendu, vous pouvez tester leur valeur et les utiliser comme auparavant.

Variables système

Voici la liste complète des **Variables système** de 4D accompagnées de leur type.

Variable	Type
OK	Entier long
Document	Texte
FldDelimit	Entier long
RecDelimit	Entier long
Error	Entier long
Error method	Texte
Error line	Entier long
Error formula	Texte
MouseDown	Entier long
KeyCode	Entier long
Modifiers	Entier long
MouseX	Entier long
MouseY	Entier long
MouseProc	Entier long

Variables des états rapides

Lorsqu'on crée une colonne calculée dans un état, 4D crée automatiquement une variable C1 pour la première, C2, C3... pour les autres. Généralement, cette création est faite de façon transparente pour vous.
Dans le cas où vous utiliseriez ces variables dans des méthodes, souvenez-vous que, comme les autres variables, les variables C1, C2... ne peuvent être retypées.

Constantes prédéfinies 4D

La liste des constantes prédéfinies de 4D peut être consultée dans ce manuel via la **Liste des thèmes de constantes**, vous pouvez également les visualiser dans l'Explorateur, en mode Développement.

Précisions de syntaxe

Le compilateur suit la syntaxe habituelle des commandes 4D et, en ce sens, ne vous demande aucun comportement particulier dans l'optique de la compilation.

Cette section propose cependant certains rappels et quelques précisions :

- Certaines commandes influant sur le type d'une variable peuvent, si vous n'y prêtez pas attention, conduire à des conflits de type.
- Certaines commandes admettant des syntaxes ou des paramètres différents, il est préférable de savoir ce qu'il est plus approprié de choisir.

Chaînes de caractères

Code de caractere (LaChaine)

Pour les routines opérant sur les chaînes, seule la fonction **Code de caractere** réclame une attention plus particulière. Lorsque vous travaillez en mode interprété, vous pouvez indifféremment passer une chaîne non vide ou vide à cette fonction.
En compilé, vous ne pouvez pas passer une chaîne vide.
Si vous le faites, le compilateur ne peut détecter une erreur à la compilation si l'argument passé à **Code de caractere** est une variable.

Communications

ENVOYER VARIABLE(LaVariable)
RECEVOIR VARIABLE(LaVariable)
Ces deux commandes permettent d'écrire et de relire des variables envoyées sur disque. On passe des variables en paramètres à ces commandes. Souvenez-vous simplement qu'il faudra toujours relire une variable d'un type dans une variable de même type. Supposons que vous souhaitiez envoyer une liste de variables dans un fichier. Afin de ne pas prendre de risque de changement de type par inattention, nous vous recommandons d'adopter une méthode de travail simple qui consiste à indiquer en début de liste le type des variables envoyées. Ainsi, lorsque vous relirez ces variables, vous commencerez toujours par récupérer cet indicateur dont vous connaissez le type. Ensuite, vous appellerez **RECEVOIR VARIABLE** en toute connaissance de cause par l'intermédiaire d'un *Au cas ou*.

Exemple :

```
REGLER SERIE(12;"LeFichier")
Si(OK=1)
   $Type:=Type([Client]CA_Cumulé)
   ENVOYER VARIABLE($Type)
   Boucle($i;1;Enregistrements trouves)
      $CA_Envoi:=[Client]CA_Cumulé
      ENVOYER VARIABLE($CA_Envoi)
   Fin de boucle
Fin de si
REGLER SERIE(11)
REGLER SERIE(13;"LeFichier")
Si(OK=1)
   RECEVOIR VARIABLE($Type)
   Au cas ou
      :($Type=Est une variable chaîne)
         RECEVOIR VARIABLE($LaChaine)
 `Traitement de la variable reçue
      :($Type=Est un numérique)
         RECEVOIR VARIABLE($LeRéel)
 `Traitement de la variable reçue
      :($Type=Est un texte)
         RECEVOIR VARIABLE($LeTexte)
 `Traitement de la variable reçue
   Fin de cas
Fin de si
REGLER SERIE(11)
```

Définition structure

Champ(Ptr_Champ) ou **(NoDeTable;NoDeChamp)**
Table(Ptr_Table) ou **(Ptr_Champ)** ou **(NoDeTable)**
Dans l'optique du compilateur, ces deux commandes ne comportent rien de spécifique. Nous appelons simplement votre attention sur un cas pratique : ces deux commandes retournent des résultats de type différent suivant le paramètre qui leur est passé :

- si vous passez un pointeur à la fonction **Table**, le résultat retourné sera de type Numérique.
- si vous passez un Numérique à la fonction **Table**, le résultat retourné sera de type Pointeur.
  On comprend aisément que ces deux fonctions ne suffisent pas au compilateur pour déterminer le type du résultat. Dans ce cas, pour qu'il n'y ait aucune ambiguïté, utilisez une directive de compilation.

Documents système

Nous rappellerons simplement que la référence d'un document renvoyée par les fonctions **Ouvrir document**, **Ajouter a document** et **Creer document** est

de type Heure.

Fonctions mathématiques

Modulo(LaValeur;Diviseur)
L'expression "25 modulo 3" peut s'écrire de deux façons différentes dans 4D :

```
LaVariable:=Modulo(25;3)
```

ou

```
LaVariable:=25%3
```

Il existe pour le compilateur une différence entre ces deux écritures : **Modulo** s'applique à tous les types de numériques tandis que l'opérateur % s'applique exclusivement aux Entiers et Entiers longs (si les opérandes de l'opérateur % dépassent les limites des Entiers longs, le résultat renvoyé sera probablement faux).

Interruptions

APPELER 4D
APPELER SUR EVENEMENT(LaMéthode {; NomProcess})
STOP

APPELER SUR EVENEMENT

Pour la gestion des interruptions, le langage de 4D dispose de la commande **APPELER 4D**. Cette commande devra être utilisée lorsque vous vous servirez de la commande **APPELER SUR EVENEMENT**.

On pourrait définir cette commande comme une directive de gestion des événements.
Seul le noyau de 4D peut détecter un événement Système (clic souris, action sur le clavier…). Dans la plupart des cas, des appels au noyau sont lancés par le code compilé lui-même, de façon transparente pour vous.

En revanche, dans le cas où vous attendez un événement sans rien faire, comme dans une boucle d'attente, il est bien évident qu'aucun appel n'est effectué.

Exemple sous Windows

```
  `Méthode projet ClicSouris
Si(MouseDown=1)
   <>vTest:=Vrai
   ALERTE("Quelqu'un a cliqué avec la souris")
Fin de si

  `Méthode projet Attente
<>vTest:=Faux
APPELER SUR EVENEMENT("ClicSouris")
Tant que(<>vTest=Faux)
  `Boucle d'attente de l'événement
Fin tant que
APPELER SUR EVENEMENT("")
```

Dans ce cas, vous ajouterez la directive **APPELER 4D** de la façon suivante :

```
  `Méthode projet Attente
<>vTest:=Faux
APPELER SUR EVENEMENT("ClicSouris")
Tant que(<>vTest=Faux)
   APPELER 4D
  `Appel au noyau pour discerner un événement
Fin tant que
APPELER SUR EVENEMENT("")
```

STOP

Cette commande ne doit être utilisée que dans des méthodes projet d'interception d'erreurs. Cette commande fonctionne comme en mode interprété, sauf dans une méthode ayant été appelée par l'une des commandes suivantes : **EXECUTER FORMULE**, **APPLIQUER A SELECTION** et **_o_APPLIQUER À SOUS SÉLECTION**. Il convient d'éviter cette situation.

Tableaux

Sept routines de 4D sont utilisées par le compilateur pour déterminer le type d'un tableau. Il s'agit de :

COPIER TABLEAU(TableauSource;TableauDestination)
SELECTION VERS TABLEAU(LeChamp;LeTableau)
TABLEAU VERS SELECTION(LeTableau; LeChamp)
SELECTION LIMITEE VERS TABLEAU(Début;Fin;LeChamp;LeTableau)
LISTE VERS TABLEAU(LaListe; LeTableau{;ItemRefs})
TABLEAU VERS LISTE(LeTableau; LaListe{;ItemRefs})
VALEURS DISTINCTES(LeChamp;LeTableau)

COPIER TABLEAU

COPIER TABLEAU admet deux paramètres de type Tableau. Lorsque le compilateur rencontre cette commande pendant le typage et que l'un des paramètres tableau n'est pas déclaré ailleurs, le compilateur déduit le type du tableau non déclaré suivant le type de celui qui l'est.

Cette déduction est faite dans les deux cas suivants :

- le tableau typé ailleurs est le premier paramètre. Le compilateur donne au second tableau le type du premier.
- le tableau déclaré est le second paramètre. Dans ce cas, le compilateur donne au premier tableau le type du second.

Le compilateur étant rigoureux sur les types, un **COPIER TABLEAU** ne peut se faire que d'un tableau d'un type vers un tableau de même type.
En conséquence, si vous souhaitez faire une copie d'un tableau d'éléments de types voisins, c'est-à-dire les Entiers, Entiers longs et Numérique ou les Textes et les Alphas ou les Alphas de longueurs différentes, vous devrez le faire élément par élément.

Imaginez que vous vouliez faire une copie d'un tableau d'Entiers vers un tableau de Numériques. Procédez comme suit :

```
$Taille:=Taille tableau(TabEntier)
TABLEAU REEL(TabRéel;$Taille)
 `Donnons la même taille au tableau de Réels qu'au tableau d'Entiers
Boucle($i;1;$Taille)
   TabRéel{$i}:=TabEntier{$i}
 `Recopions chacun des éléments
Fin de boucle
```

Souvenez-vous que vous ne pouvez changer le nombre de dimensions d'un même tableau en cours de route. Vous provoqueriez une erreur de typage en copiant un tableau à une dimension dans un tableau à deux dimensions.

SELECTION VERS TABLEAU, TABLEAU VERS SELECTION, VALEURS DISTINCTES, SELECTION LIMITEE VERS TABLEAU

De même que pour 4D en mode interprété, ces quatre commandes n'exigent pas de déclaration de tableau. Le tableau non déclaré recevra le même type que le champ spécifié dans la commande.
Si vous écrivez :

```
SELECTION VERS TABLEAU([LaTable]ChampEntier;LeTableau)
```

le type de LeTableau sera donc Tableau d'Entiers à une dimension.

Dans le cas où le tableau a déjà été déclaré, veillez à ce que les champs soient du même type. Bien qu'Entier, Entier long et Réel soient des types voisins, vous ne pouvez pas supposer qu'il soient équivalents.
Vous avez en revanche plus de latitude lorsqu'il s'agit des types Texte et Alpha. Par défaut, lorsqu'un tableau n'a pas été déclaré au préalable et que vous appliquez l'une de ces commandes avec pour paramètre un champ de type Alpha, le type attribué au tableau sera Texte. Si le tableau a été déclaré auparavant comme étant de type Alpha ou de type Texte, ces commandes respecteront vos directives.

Il en est de même dans le cas des champs de type Texte : vos directives sont prioritaires.
Souvenez-vous que les commandes **SELECTION VERS TABLEAU**, **SELECTION LIMITEE VERS TABLEAU**, **TABLEAU VERS SELECTION** et **VALEURS DISTINCTES** ne s'appliquent qu'à des tableaux à une dimension.

La commande **SELECTION VERS TABLEAU** admet aussi une seconde syntaxe :
SELECTION VERS TABLEAU(LaTable;LeTableau)
Dans ce cas, la variable LeTableau sera de type Tableau d'Entiers longs. Il en est de même pour la commande **SELECTION LIMITEE VERS TABLEAU**.

LISTE VERS TABLEAU (anc. ENUMERATION VERS TABLEAU), TABLEAU VERS LISTE (anc. TABLEAU VERS ENUMERATION)

Les commandes **LISTE VERS TABLEAU** et **TABLEAU VERS LISTE** ne concernent que deux types de tableaux :

- les tableaux Alpha à une dimension,
- les tableaux Texte à une dimension.
  Ces deux commandes n'exigent pas la déclaration du tableau qui leur est passé en paramètre. Par défaut, un tableau non déclaré recevra le type Texte. Si le tableau a été déclaré auparavant comme étant de type Alpha ou de type Texte, ces commandes respecteront vos directives.

Utilisation des pointeurs dans les commandes s'appliquant aux tableaux

Le compilateur ne peut pas, durant le typage ou la compilation, détecter un conflit de type dans le cas où vous utiliseriez des pointeurs dépointés comme paramètre de commande de déclaration d'un tableau. Si vous écrivez :

```
SELECTION VERS TABLEAU([LaTable]LeChamp;LePointeur->)
```

où LePointeur-> représente un tableau, le compilateur ne peut vérifier que le type du champ et du tableau sont identiques. Il vous appartient d'éviter alors ce type de conflit.

Pour cela, le compilateur vous fournit une aide précieuse : les **Warnings**. Chaque fois qu'il rencontre une routine de déclaration de tableau dans laquelle l'un des paramètres est un pointeur, il vous délivre un message vous invitant à la vigilance.

Tableaux locaux

Si vous souhaitez compiler une base de données qui utilise des tableaux locaux (tableaux visibles uniquement par les méthodes qui les ont créés), il est nécessaire de les déclarer explicitement dans 4D avant de les utiliser.

La déclaration explicite d'un tableau signifie l'utilisation d'une commande de type **TABLEAU REEL**, **TABLEAU ENTIER**, etc.

Par exemple, si une méthode génère un tableau local d'entiers contenant 10 valeurs, vous devez écrire au préalable la ligne suivante :

```
TABLEAU ENTIER($MonTableau;10)
```

Langage

Pointeur vers(NomVariable)
Type(Objet)
EXECUTER FORMULE(Instruction)
TRACE
PAS DE TRACE

Pointeur vers

Pointeur vers est une fonction qui retourne un pointeur sur le paramètre que vous lui avez passé. Supposons que vous vouliez initialiser un tableau de pointeurs. Chacun des éléments de ce tableau pointe vers une variable donnée. Ces variables sont au nombre de douze et nous les appelons V1, V2, … V12. Vous pourriez écrire :

```
TABLEAU POINTEUR(Tab;12)
Tab{1}:=->V1
Tab{2}:=->V2

Tab{12}:=->V12
```

Vous pouvez aussi écrire :

```
TABLEAU POINTEUR(Tab;12)
Boucle($i;1;12)
   Tab{$i}:=Pointeur vers("V"+Chaine($i))
Fin de boucle
```

A la fin de l'exécution de cette séquence, vous récupérez un tableau de pointeurs dont chaque élément pointe sur une variable Vi.

Ces deux séquences sont bien entendu compilables. Toutefois, si les variables V1 à V12 ne sont pas utilisées explicitement ailleurs dans la base, le compilateur ne pourra pas les typer. Il vous faut donc les nommer ailleurs explicitement.
Cette déclaration explicite peut se faire de deux façons :

- soit en déclarant V1, V2, …V12 par une directive de compilation :

```
C_ENTIER LONG(V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12)
```

- soit en affectant ces variables dans une méthode :

```
V1:=0
V2:=0

V12:=0
```

Type (Objet)

Chaque variable de la base compilée n'ayant qu'un seul type, la fonction **Type** peut sembler d'un intérêt limité. Elle est cependant très précieuse lorsqu'on travaille avec des pointeurs. En effet, il peut être nécessaire de connaître le type de la variable pointée par un pointeur, la souplesse des pointeurs faisant que l'on ne sait pas toujours sur quoi ils pointent.

EXECUTER FORMULE

La commande **EXECUTER FORMULE**, historique dans 4D, présente des avantages en mode interprété qui n'ont pas grand sens en mode compilé. En effet, en mode compilé, le nom de la méthode passé en paramètre à cette commande sera simplement interprété. Vous ne bénéficierez donc pas de tous les avantages apportés par le compilateur, sans compter que la syntaxe de votre paramètre ne pourra pas être vérifiée.
De surcroît, vous ne pouvez pas lui passer des variables locales comme paramètres.
On peut avantageusement remplacer un **EXECUTER FORMULE** par une série d'instructions. Nous vous en donnons ici deux exemples.

Soit la séquence suivante :

```
i:=FctFormulaire
EXECUTER FORMULE("FORM FIXER ENTREE (Formulaire"+Chaine(i)+")")
```

Elle peut être remplacée par :

```
i:=FctFormulaire
VarFormulaire:="Formulaire"+Chaine(i)
FORM FIXER ENTREE(VarFormulaire)
```

Examinons maintenant un deuxième cas :

```
$Num:=ChoixImprimante
EXECUTER FORMULE("Impri"+$Num)
```

Ici, l'**EXECUTER FORMULE** peut être facilement remplacé par un *Au cas ou* :

```
Au cas ou
   :($Num=1)
      Impri1
   :($Num=2)
      Impri2
   :($Num=3)
      Impri3
Fin de cas
```

La commande **EXECUTER FORMULE**peut toujours être avantageusement remplacée. En effet, la méthode à exécuter est prise dans la liste des méthodes projet de la base ou des commandes 4D, qui sont en nombre limité. En conséquence, il est toujours possible de remplacer la commande **EXECUTER FORMULE**soit par un *Au cas ou*, soit par une autre commande.

TRACE, PAS DE TRACE

Ces deux commandes, précieuses en mode interprété, n'ont pas de fonction en mode compilé. Vous pouvez cependant les laisser dans vos méthodes : **TRACE** et **PAS DE TRACE** seront simplement ignorées par le compilateur.

Variables

Indefinie(LaVariable)
ECRIRE VARIABLES(NomduDoc;Variable1{; Variable2…})
LIRE VARIABLES(NomduDoc;Variable1{; Variable2…})
EFFACER VARIABLE(LaVariable)

Indefinie

Compte tenu du processus de typage par le compilateur, une variable ne peut à aucun moment être indéfinie en mode compilé. En effet, toutes les variables ont une existence dès la fin de la compilation. La fonction **Indefinie** retourne donc toujours **Faux**, quel que soit le paramètre que vous lui passez.

Note : Pour savoir si une application tourne en mode compilé, utilisez la fonction **Mode compile**.

ECRIRE VARIABLES, LIRE VARIABLES

En mode interprété, après l'exécution d'un **LIRE VARIABLES**, vous pouvez savoir si le document existe en testant si l'une des variables, théoriquement lue, est indéfinie ou non. Ceci n'est bien évidemment plus possible avec le compilateur, puisque la fonction **Indefinie** renvoie toujours **Faux**.

La méthode la plus simple pour réaliser cette opération en mode interprété comme en mode compilé est la suivante :
1. Initialisez les variables que vous allez recevoir à une valeur dont vous êtes sûr qu'elles ne sont pas initialisées.
2. Après le **LIRE VARIABLES**, comparez l'une des variables reçues à la valeur d'initialisation.
La méthode s'écrira alors de la façon suivante :

```
Var1:="xxxxxx"
 `"xxxxxx" est une valeur qui ne peut pas être ramenée par un LIRE VARIABLES
Var2:="xxxxxx"
Var3:="xxxxxx"
Var4:="xxxxxx"
LIRE VARIABLES("LeDocument";Var1;Var2;Var3;Var4)
Si(Var1="xxxxxx")
 `Le document n'a pas été trouvé

Sinon
 `Le document a été trouvé

Fin de si
```

EFFACER VARIABLE

Cette routine admet deux syntaxes différentes en mode interprété :
EFFACER VARIABLE(LaVariable)
EFFACER VARIABLE("a")
En mode compilé, la première syntaxe, **EFFACER VARIABLE(LaVariable)**, provoque non la destruction de la variable, mais sa réinitialisation (remise à zéro pour un numérique, chaîne vide pour une chaîne de caractères ou un texte…), aucune variable ne pouvant être indéfinie en mode compilé.
En conséquence, **EFFACER VARIABLE** ne libère pas de mémoire en mode compilé sauf dans quatre cas : les variables de type Texte, Image, BLOB et les tableaux.
Pour un tableau, **EFFACER VARIABLE** équivaut à une nouvelle déclaration du tableau ou les tailles sont remises à zéro.

Pour un tableau LeTableau dont les éléments sont de type *Entier*, **EFFACER VARIABLE(LeTableau)** équivaut à l'une des expressions suivantes :

```
TABLEAU ENTIER(LeTableau;0)
 `s'il s'agit d'un tableau à une dimension
TABLEAU ENTIER(LeTableau;0;0)
 `s'il s'agit d'un tableau à deux dimensions
```

La seconde syntaxe, **EFFACER VARIABLE("a")**, n'est pas compatible avec le compilateur puisque celui-ci accède aux variables non par leur nom, mais par leur adresse mémoire.

Précisions concernant l'utilisation de pointeurs

Précisions concernant l'utilisation de pointeurs

Les commandes suivantes ont un point commun : elles admettent toutes un premier paramètre [LaTable] facultatif alors que le second paramètre peut être un pointeur.

ADJOINDRE ELEMENT	FORM FIXER ENTREE
ALLER A ENREGISTREMENT	FORM FIXER SORTIE
ALLER DANS SELECTION	_o_GRAPHE SUR SELECTION
APPLIQUER A SELECTION	IMPRIMER ETIQUETTES
CHARGER ENSEMBLE	Imprimer ligne
CHERCHER	IMPORTER TEXTE
CHERCHER DANS SELECTION	LECTURE DIF
CHERCHER PAR FORMULE	LECTURE SYLK
CHERCHER PAR FORMULE DANS SELECTION	LIEN RETOUR
COPIER SELECTION	NOMMER ENSEMBLE
DEPLACER SELECTION	REDUIRE SELECTION
DIALOGUE	ENLEVER ELEMENT
EXPORTER TEXTE	TRIER
ECRITURE DIF	TRIER PAR FORMULE
ECRITURE SYLK	UTILISER PARAMETRES IMPRESSION
ENSEMBLE VIDE	VERROUILLE PAR
QR ETAT	

Il est parfaitement possible en mode compilé de garder ce caractère optionnel au paramètre [LaTable]. Le compilateur fait toutefois une supposition dans le cas où le premier paramètre passé à l'une de ces commandes est un pointeur. Ne pouvant pas savoir sur quoi pointe ce pointeur, il suppose qu'il s'agit d'un pointeur de Table.

Prenons par exemple le cas de la commande **CHERCHER** dont la syntaxe est la suivante :
CHERCHER({LaTable{;LaFormule{;*}}})
Le premier élément du paramètre *LaFormule* doit être un champ.
Si vous écrivez :

```
CHERCHER(LePtrChamp->=Vrai)
```

le compilateur va chercher en deuxième élément de formule un symbole représentant un champ. Or, il trouvera le signe "=". Il délivrera un message d'erreur, ne pouvant identifier la commande avec une expression qu'il sait traiter.

En revanche, si vous écrivez :

```
CHERCHER(LePtrTable->;LePtrChamp->=Vrai)
```

ou

```
CHERCHER([LaTable];LePtrChamp->=Vrai)
```

vous levez toute ambiguïté.

Utilisation directe de commandes retournant des pointeurs

Les commandes dont le premier paramètre [laTable] est facultatif et dont le second paramètre est également facultatif présentent une particularité lors de l'utilisation de pointeurs. Dans ce contexte, pour des raisons internes l'utilisation en tant que paramètre d'une commande retournant un pointeur (par exemple **Table du formulaire courant**) n'est pas autorisée par le compilateur (une erreur est générée).

C'est le cas, par exemple, de la commande **FORM CAPTURE ECRAN**. Le code suivant fonctionnera en mode interprété mais sera rejeté en cas de compilation :

```
 //provoque une erreur à la compilation
FORM CAPTURE ECRAN(Table du formulaire courant->;$nomForm;$monImage)
```

Dans ce cas, il suffit d'utiliser une variable intermédiaire afin que le même code soit validé par le compilateur :

```
 //code équivalent compilable
C_POINTEUR($ptr)
$ptr:=Table du formulaire courant
FORM CAPTURE ECRAN($ptr->;$nomForm;$monImage)
```

Constantes

Si vous créez vos propres ressources 4DK# (constantes), assurez-vous que les numériques soient de type Entier long (L) ou Réel (R) et les alphas de type Chaîne (S). Tout autre type génèrera un Warning.

Conseils d'optimisation

Il est difficile, voire impossible, de consigner une fois pour toutes des instructions pour "bien programmer". Tout au plus pouvons-nous renouveler les recommandations générales vous invitant à bien structurer vos programmes, grâce notamment aux possibilités de programmation générique offertes par 4D.
De fait, il est clair que si vous essayez de compiler une base très bien structurée, vous obtiendrez de bien meilleurs résultats que si votre base est mal structurée. Par exemple, si vous écrivez une méthode projet générique qui gère n méthodes objet, vous obtiendrez de bien meilleurs résultats, en interprété comme en compilé, que si les n méthodes objet comportent n fois les mêmes séquences d'instructions.
La qualité de votre programmation peut donc avoir des effets sur la qualité du code compilé.
Si vous débutez, vous améliorerez progressivement votre code 4D grâce à l'habitude. De la même façon, c'est en utilisant fréquemment le compilateur que vous acquerrez les réflexes qui permettent inconsciemment d'aller droit au but.
En attendant cependant, voici quelques conseils et astuces qui vous permettront de gagner du temps dans des opérations simples et fréquentes.

Remarque préliminaire : les commentaires

Certaines astuces que nous vous conseillons peuvent rendre votre code moins lisible par une personne extérieure ou par vous-même ultérieurement. En conséquence, n'hésitez pas à assortir vos méthodes de commentaires détaillés. En effet, si les commentaires peuvent parfois ralentir une base interprétée, ils n'ont strictement aucune influence sur les temps d'exécution d'une base compilée.

Optimisation par les directives de compilation

Les directives de compilation peuvent vous aider à accélérer considérablement votre code.
Par défaut, le compilateur donne le type le plus large possible à vos variables afin de ne pas vous pénaliser. Si vous ne typez pas par une directive de compilation une variable désignée par l'instruction **Var:= 5**, le compilateur lui donnera le type Réel, même s'il s'agit de l'affectation d'une valeur entière.
Nous allons maintenant voir comment, dans certains cas, déclarer une variable peut vous faire gagner beaucoup de temps.

Réels

Par défaut, le compilateur donne le type Réel aux variables numériques non typées par directive de compilation et si les Propriétés de la base n'indiquent pas le contraire. Or, les calculs sur un Réel sont plus lents que sur un Entier. Lorsque vous êtes sûr qu'un de vos numériques s'exprimera toujours dans votre base sous forme d'un Entier, il est avantageux de le déclarer par la directive de compilation **C_ENTIER LONG**.
Une bonne habitude à prendre, par exemple, est de systématiquement déclarer vos compteurs de boucles comme Entier par directive de compilation.
Par ailleurs, certaines fonctions standard de 4D renvoient des Entiers (exemple : **Code de caractere**, **Ent**...). Si vous affectez le résultat d'une de ces fonctions à une variable non typée de votre base, le compilateur lui donnera le type Réel et non Entier. Pensez donc à déclarer ces variables par des directives de compilation si vous êtes certain qu'elles ne seront utilisées que dans ces conditions.
Prenons un exemple simple. Une fonction renvoyant une valeur aléatoire comprise entre deux bornes peut s'écrire :

```
$0:=Modulo(Hasard;($2-$1+1))+$1
```

Elle renvoie toujours une valeur entière. Si cette fonction est écrite ainsi, le compilateur donnera à $0 le type Réel et non le type Entier ou Entier long. Il est donc préférable d'écrire cette méthode sous la forme :

```
C_ENTIER LONG($0)
$0:=Modulo(Hasard;($2-$1+1))+$1
```

Le paramètre rendu par la méthode sera plus petit et il sera donc plus rapide de faire appel à cette méthode.
Prenons un autre exemple simple. Supposons que vous ayez déclaré deux variables en Entier long par l'instruction suivante :

```
C_ENTIER LONG($var1;$var2)
```

et qu'une troisième variable non typée reçoive la somme des deux autres :

```
$var3:=$var1+$var2.
```

Il vous faudra explicitement déclarer $var3 comme Entier long si vous voulez effectivement que $var3 soit de type Entier long, sinon le compilateur la typera par défaut en Réel.
Note : Attention au mode de calcul dans 4D. En mode compilé, ce n'est pas le type de la variable recevant le calcul qui détermine le mode de calcul, mais le type des opérandes. Dans l'exemple ci-dessous, le calcul s'effectue sur des Entiers longs :

```
C_REEL($var3)
C_ENTIER LONG($var1;$var2)
$var1:=2147483647
$var2:=1
$var3:=$var1+$var2
```

$var3 prendra la valeur –2147483648 en mode compilé comme en interprété. Dans l'exemple suivant :

```
C_REEL($var3)
C_ENTIER LONG($var1)
$var1:=2147483647
$var3:=$var1+1
```

pour des raisons d'optimisation, le compilateur considère la valeur 1 comme une valeur entière. $var3 prendra alors la valeur -2147483648 en mode compilé (car le calcul s'effectue sur des Entiers longs) et 2147483648 en mode interprété (car le calcul s'effectue sur des Réels).

Les boutons sont un cas particulier de Réel qu'il est possible de déclarer en Entier long.

Chaînes de caractères

Si vous souhaitez tester la valeur d'un caractère, il est plus rapide de faire la comparaison sur son **Code de caractere** que sur le caractère lui-même. En effet, la routine standard de comparaison de caractères prend en compte toutes les variations du caractère, comme par exemple les accentuations.

Remarques diverses

Tableaux à deux dimensions

Les traitements sur les tableaux à deux dimensions seront mieux gérés si la deuxième dimension est plus grande que la première.

Par exemple, un tableau déclaré sous la forme :

```
TABLEAU ENTIER(LeTableau;5;1000)
```

sera mieux géré qu'un tableau déclaré sous la forme :

```
TABLEAU ENTIER(LeTableau;1000;5)
```

Champs

Vous gagnerez du temps lorsque, si vous devez effectuer plusieurs calculs sur un champ, vous rangez la valeur de ce champ dans une variable et que vous effectuez vos calculs sur cette variable, plutôt que de faire directement les calculs sur le champ. Illustrons notre propos par un exemple. Prenons la méthode suivante :

```
Au cas ou
   :([Client]Dest="Paris")
      Transport:="Coursier"
   :([Client]Dest="Corse")
      Transport:="Avion"
   :([Client]Dest="Etranger")
      Transport:="Service express"
   Sinon
      Transport:="Poste"
Fin de cas
```

La séquence ci-dessus sera plus lente à exécuter que si elle avait été écrite de la façon suivante :

```
$Dest:=[Client]Dest
Au cas ou
   :($Dest="Paris")
      Transport:="Coursier"
   :($Dest="Corse")
      Transport:="Avion"
   :($Dest="Etranger")
      Transport:="Service express"
   Sinon
      Transport:="Poste"
Fin de cas
```

Pointeurs

De même que pour les champs, il est plus rapide de travailler sur une variable intermédiaire que sur un pointeur dépointé. Vous gagnerez énormément de temps si, lorsque vous devez faire plusieurs calculs sur une variable pointée, vous rangez le pointeur dépointé dans une variable.

Vous disposez par exemple d'un pointeur pointant sur un champ ou sur une variable, MonPtr. Ensuite, vous souhaitez faire une série de tests sur la valeur pointée par MonPtr. Vous pouvez écrire :

```
Au cas ou
   :(MonPtr->=1)
      Séquence 1
   :(MonPtr->=2)
      Séquence 2

Fin de cas
```

Il est plus rapide d'exécuter cette série de tests si elle est écrite ainsi :

```
Temp:=MonPtr->
Au cas ou
   :(Temp=1)
      Séquence 1
   :(Temp=2)
      Séquence 2

Fin de cas
```

Variables locales

Utiliser des variables locales permet, dans bien des cas, de mieux structurer son code. Ces variables présentent d'autres avantages :

- Les variables locales prennent moins de place en mémoire lors de l'utilisation d'une base : en effet, elles sont créées lorsqu'on entre dans la méthode où elles sont utilisées et détruites dès qu'on sort de cette méthode.
- Le code généré est optimisé pour les variables locales, en particulier de type Entier long. Pensez-y pour vos compteurs de boucle.

Messages d'erreurs

Cette section détaille les principaux messages délivrés par le compilateur. Ces messages sont de plusieurs types :

- les warnings, qui vous aident à déjouer des pièges facilement évitables ;
- les erreurs, qu'il vous appartient de corriger ;
- les messages de contrôle d'exécution, délivrés dans 4D.

Les warnings

Ces messages sont délivrés tout au long du processus de compilation. Chaque message est accompagné ici d'un exemple de ce qui a pu le provoquer et, éventuellement, d'une explication supplémentaire.

Utilisation de pointeur(s) comme paramètre(s) de **COPIER TABLEAU**

```
COPIER TABLEAU(LePointeur->;LeTableau)
```

Utilisation de pointeur(s) comme paramètre(s) de **SELECTION VERS TABLEAU**

```
SELECTION VERS TABLEAU(LePointeur->;LeTableau)
SELECTION VERS TABLEAU([LaTable]LeChamp;LePointeur->)
```

Utilisation de pointeur(s) comme paramètre(s) de **TABLEAU VERS SELECTION**

```
TABLEAU VERS SELECTION(LePointeur->;[LaTable]LeChamp)
```

Utilisation de pointeur(s) comme paramètre(s) de **LISTE VERS TABLEAU**

```
LISTE VERS TABLEAU(Enum;LePointeur->)</gen9>
```

Utilisation de pointeur(s) comme paramètre(s) de **TABLEAU VERS LISTE**

```
TABLEAU VERS LISTE(LePointeur->;Enum)
```

Utilisation de pointeur(s) dans une déclaration de tableau

```
TABLEAU REEL(LePointeur->;5)
```

L'instruction **TABLEAU REEL(LeTableau;LePointeur->)** ne provoquera pas cet avertissement. La valeur de la dimension d'un tableau n'a pas d'influence sur son type.

Utilisation de pointeur(s) comme paramètre(s) de **VALEURS DISTINCTES**

```
VALEURS DISTINCTES(LePointeur->;LeTableau)
```

Utilisation de la fonction **Indefinie**

```
Si(Indefinie(LaVariable))
```

Cette méthode est protégée par un mot de passe.

Le Formulaire LeFormulaire contient un bouton avec action sans nom dans la page 1.
Tous vos boutons avec action doivent avoir un nom afin d'éviter des conflits.

Le pointeur utilisé dans cette commande doit pointer sur un Alphanumérique.

```
LePointeur->≤2≥:="a"
```

Le pointeur utilisé dans cette commande doit pointer sur un Entier, un Entier long ou un NumériqueLaChaine≤LePointeur->≥:="a"

Un indice de tableau doit être Entier, Entier long ou Numérique.

```
ALERTE(LeTableau{LePointeur->})
```

Il manque un paramètre à l'appel de la commande du plug-in.

```
WR FIXER POLICE(LaZone)
```

Note : Il est possible de désactiver et d'activer individuellement des warnings avec les balises suivantes :
//%W-numero_de_warning pour désactiver un warning
//%W+numero_de_warning pour activer un warning
Ces activations et désactivations sont effectives pour tout le code analysé dans la suite du plan de compilation. Si on veut désactiver ou activer un warning de manière globale, il suffit d'insérer la balise idoine dans une méthode nommée "Compiler_xxx" car ces méthodes sont analysées en premier par le compilateur. Par exemple, pour désactiver le warning "Utilisation de pointeur(s) comme paramètre(s) de COPIER TABLEAU", il suffit d'insérer la balise "//%W-518.1" à l'endroit désiré.

Les erreurs

Ces messages vous sont délivrés tout au long du processus de compilation. Il vous appartient de corriger ces erreurs afin de permettre au compilateur de générer une base compilée. Chaque message est accompagné ici d'un exemple de ce qui a pu le provoquer et éventuellement, d'une explication supplémentaire.

Typage

Cet opérateur ne peut s'appliquer à ce type de variable. Cette affectation provoquerait un conflit de type.

```
LeRéel:=12,3
LeBooléen:=Vrai
LeRéel:=LeBooléen
```

Changement du nombre de dimensions d'un tableau.

```
TABLEAU TEXTE(LeTableau;5;5)
TABLEAU TEXTE(LeTableau;5)
```

Conflit de type sur la variable LeTableau dans le formulaire LeFormulaire.

```
TABLEAU ENTIER(LeTableau)
```

Déclaration d'un tableau sans indice.

```
TABLEAU ENTIER(LeTableau)
```

Il manque une variable.

```
COPIER TABLEAU(LeTableau;"")
```

Impossible de déterminer le type de LaVariable. Cette variable est utilisée dans la méthode M1.
Le type de LaVariable n'a pu être déterminé. Une directive de compilation est nécessaire.

Type de constante invalide

```
OK:="Il fait beau"
```

La méthode M1 est inconnue.
La ligne contient un appel à une méthode qui n'existe pas ou plus.

Le champ utilisé dans cette expression provoque un conflit de type.

```
MaDate:=Ajouter a date(LeChampBool;1;1;1)
```

La variable LaVariable n'est pas une méthode.

```
LaVariable(1)
```

La variable LaVariable n'est pas un tableau.

```
LaVariable{5}:=12
```

Le résultat de la fonction est incompatible avec l'expression.

```
LeTexte:="Numéro"+Num(i)
```

Les types utilisés dans l'expression sont incompatibles.

```
LEntier:=LaDate*LeTexte
```

Utilisation de la variable $i de type alphanumérique comme variable de type réel.

```
$i:="3"
$($i):=5
```

L'indice du tableau n'est pas numérique.

```
TabEntier{"3"}:=4
```

Retypage de la variable LaVariable du type Texte en tableau de type Texte.

```
C_TEXTE(LaVariable)
COPIER TABLEAU(TabTexte;LaVariable)
```

Retypage de la variable LaVariable du type Texte en type Réel.

```
LaVariable:=Num(LaVariable)
```

Retypage du tableau LeBooléen du type Booléen en variable de type Réel.

```
LaVariable:=LeBooléen
```

Retypage du tableau du type Entier en type tableau de type Texte.

```
TABLEAU TEXTE(TabEntier;12)
```

si TabEntier a été déclaré ailleurs comme tableau d'Entiers.

Seuls les pointeurs peuvent être suivis du signe ->

```
LaVariable->:=5
```

si LaVariable n'est pas de type Pointeur.

Utilisation de la variable LaVar1 de type Texte comme une variable de type Numérique.

```
LaVar1:=3,5
```

Utilisation d'un champ de type incorrect. [LaTable]LeChamp est un champ de type Date.

LaVariable est de type Numérique.

```
LaVariable:=[LaTable]LeChamp
```

Syntaxe

Cette fonction ne retourne pas un pointeur.

```
LaVariable:=Num("Il fait beau")->
```

Il n'est pas possible de dépointer cette fonction.

Erreur de syntaxe.

```
Si(LeBooléen)
Fin de boucle
```

Cette expression contient trop d'accolades ouvrantes ({).
La ligne comporte plus d'accolades ouvrantes que d'accolades fermantes.

Cette expression contient trop d'accolades fermantes (}).
La ligne comporte plus d'accolades fermantes que d'accolades ouvrantes.

Il manque une parenthèse fermante.
La ligne comporte plus de parenthèses ouvrantes que de fermantes.

Il manque une parenthèse ouvrante.
La ligne comporte plus de parenthèses fermantes que d'ouvrantes.

J'attendais un champ.

```
Si(Modifie(LaVariable))
```

Il manque une accolade ouvrante.

```
C_ENTIER($
```

J'attendais une variable.

```
C_ENTIER([LaTable]LeChamp)
```

J'attendais un nombre constant.

```
C_ENTIER(${"3"})
```

J'attendais un point virgule.

```
COPIER TABLEAU(LeTableau1 LeTableau2)
```

Cette expression contient trop d'indices de chaîne ouvrants (MacOS).

```
LaChaine≤3:="a"
```

Cette expression contient trop d'indices de chaîne fermants (MacOS).

```
LaChaine3≥:="a"
```

Cette expression contient trop d'indices de chaîne ouvrants (Windows).

```
LaChaine[[3:="a"
```

Cette expression contient trop d'indices de chaîne fermants (Windows).

```
LaChaine 3]]:="a"
```

Je n'attendais pas un champ de type sous-table

```
TABLEAU VERS SELECTION(LeTableau;Soustable)
```

Le type du paramètre de SI doit être booléen.

```
Si(LePointeur)
```

L'expression est trop complexe.
Divisez votre ligne en plusieurs sous-opérations.

Méthode trop complexe.
Trop d'imbrications de Au cas ou...Fin de cas et de Si…Fin de si.

Référence à un champ inconnu.
Votre méthode, probablement copiée d'une autre base, contient •???• à la place d'un nom de champ.

Référence à une table inconnue.
Votre méthode, probablement copiée d'une autre base, contient •???• à la place d'un nom de table.

Un pointeur ne peut être défini sur cette expression.

```
LePointeur:=->LaVariable+3
```

Utilisation incorrecte des indices de chaînes de caractères.

```
LeNumérique≤3≥ou LeNumérique[[3]]
```

ou bien

```
LaChaine≤LaVariable≥ou LaChaine[[LaVariable]]
```

où LaVariable n'est pas une variable Numérique.

Paramètres

Ce résultat de fonction ne peut pas être paramètre de cette méthode.

```
LaMéthode(Num(LaChaine))
```

si LaMéthode attend un paramètre de type Booléen.

Cette routine reçoit trop de paramètres

```
TABLE PAR DEFAUT(LaTable;LeFormulaire)
```

Cette valeur ne peut pas être paramètre de cette méthode.

```
LaMéthode(3+2)
```

si LaMéthode attend un paramètre de type Booléen.

Conflit de type sur la variable $0.

```
C_ENTIER($0)
$0:=Faux
```

Conflit de type sur le paramètre générique.

```
C_ENTIER(${3})
Boucle($i;3;5)
   ${$i}:=Chaine($i)
Fin de boucle
```

La routine n'attend pas de paramètre.

```
AFFICHER BARRE OUTILS(MaVar)
```

La routine nécessite au moins un paramètre.

```
TABLE PAR DEFAUT
```

La variable LaChaine ne peut pas être paramètre de cette méthode.

```
LaMéthode(LaChaine)
```

si LaMéthode attend un paramètre de type Booléen.

Le type du paramètre $1 dans la méthode est différent de celui du paramètre à l'appel.

```
Calcul("3+2")
```

avec la directive **_o_C_ENTIER($1)** dans Calcul.

Le type du paramètre passé ne correspond pas au type du paramètre attendu.

```
Impri("LaserWriter")
```

si dans la méthode Impri, $1 est de type Numérique.

L'un des paramètres de COPIER TABLEAU est une variable.

```
COPIER TABLEAU(LaVariable;LeTableau)
```

Retypage du paramètre $1 du type réel en type Texte.

```
$1:=Chaine($1)
```

Un paramètre ne peut être un tableau.

```
RéInit(LeTableau)
```

Pour passer un tableau à une méthode, il faut passer un pointeur sur ce tableau.

Un paramètre ne peut être utilisé dans l'appel de cette routine.

```
RECEVOIR VARIABLE($1)
```

Utilisation du paramètre $1 de type Booléen comme une variable de type Entier.

```
LIRE PROPRIETES CHAMP(NoDeTable;NoDeChamp;Type;$1)
```

Opérateurs

Cet opérateur ne peut s'appliquer à ce type de variable.

```
LeBool2:=LeBool1+Vrai
```

L'addition ne peut pas s'appliquer à des Booléens.

Je n'attendais pas l'opérateur >

```
CHERCHER(LaTable;[LaTable]LeChamp=0;>)
```

Les deux opérandes ne sont pas comparables.

```
Si(LeLongE=Image2)
```

Le signe moins ne peut pas être utilisé dans cette expression.

```
LeBool:=-Faux
```

Plug-ins

La commande PExt du plug-in est mal définie.

Il manque des paramètres à l'appel de la commande du plug-in.

Le nombre de paramètres envoyés à la commande du plug-in est trop grand.

La commande LaVariable du plug-in est mal définie.

Erreurs générales

Deux méthodes portent le même nom : LeNom.
Pour pouvoir compiler votre base, il faut que toutes les méthodes projets aient des noms différents.

Erreur interne n° xx.
Au cas où ce message apparaîtrait dans l'une de vos bases, téléphonez au support technique de 4D et signalez le numéro de l'erreur.

Je n'ai pas pu déterminer le type de LaVariable. Cette variable est utilisée dans la méthode M1.
Le type de LaVariable n'a pu être déterminé. Une directive de compilation est nécessaire.

La méthode originale est endommagée.
La méthode est endommagée dans la structure originale. Supprimez-la ou remplacez-la.

Méthode 4D inconnue.
La méthode est endommagée.

Retypage de la variable LaVariable dans le Formulaire LeFormulaire.
Ce message apparaîtra si vous donnez, par exemple, le nom OK à une variable de type Graphe dans un formulaire.

Une fonction et une variable portent le même nom : LeNom.
Renommez soit la fonction, soit la variable.

Une variable dessinée dans le Formulaire LeFormulaire a le même nom qu'une fonction : LeNom.
Renommez soit la fonction, soit la variable.

Une méthode et une variable portent le même nom : LeNom.
Renommez soit la méthode, soit la variable.

Une commande du plug-in et une variable portent le même nom : LeNom.
Renommez soit la commande du plug-in, soit la variable.

Une commande qui n'est pas thread-safe ne peut pas être appelée par une méthode déclarée thread-safe.
Modifiez la méthode afin qu'elle devienne thread-safe (ne pas utiliser de commande non thread-safe) ou changez la propriété de déclaration de la méthode

pour lancer un process coopératif
La méthode 'NomMethode' qui n'est pas thread-safe ne peut pas être appelée par une méthode déclarée thread-safe.
Modifiez la méthode afin qu'elle devienne thread-safe ou changez la propriété de déclaration de la méthode pour lancer un process coopératif

Les messages du contrôle d'exécution

Ces messages sont délivrés dans 4D, lors de l'utilisation de la base compilée. Ils sont affichés dans 4D dans une fenêtre d'erreur spécifique.

Dépassement de capacité d'un tableau.
Si LeTableau est un tableau à 5 éléments à un instant donné, ce message apparaîtra si vous essayez d'accéder à l'élément LeTableau{17} à cet instant.

Division par zéro.

```
Var1:=0
Var2:=2
Var3:=Var2/Var1
```

Le paramètre utilisé n'a pas été passé.
Utilisation de la variable $4 alors que seuls trois paramètres ont été passés lors de l'appel courant.

Le pointeur n'est pas correctement initialisé.

```
LePointeur->:=5
```

si LePointeur n'a pas encore été initialisé.

La chaîne dans laquelle se fait l'affectation est trop courte.

```
C_ALPHA(LaChaine1;5)
C_ALPHA(LaChaine2;10)
LaChaine2:="Bonjour"
LaChaine1:=LaChaine2
```

L'indice de chaîne n'est pas valide (trop grand ou négatif).

```
i:=-30
LaChaine≤i≥:=LaChaine2 ou LaChaine[[i]]:=LaChaine2
```

La chaîne passée en paramètre est vide ou non initialisée.

```
LaChaine≤1≥:=""
LaChaine[[1]]:=""
```

Modulo par zéro.

```
Var1:=0
Var2:=2
Var3:=Var2% Var1
```

Paramètres incorrects dans une commande EXECUTER FORMULE.

```
EXECUTER FORMULE("MaMéthode(MonAlpha)")
```

si MaMéthode attend un paramètre autre qu'alphanumérique.

Pointeur sur une variable inconnue du compilateur.

```
LePointeur:=Pointeur vers("LaVariable")
LePointeur:="MaChaîne"
```

si LaVariable n'apparaît pas explicitement dans la base.

Tentative de retypage par l'intermédiaire d'un pointeur.

```
LeBooleen:=LePointeur->
```

si LePointeur référence un champ de type Entier.

Utilisation incorrecte d'un pointeur.

```
LeCaractère:=LaChaine≤LePointeur->≥
LeCaractère:=LaChaine[[LePointeur]]
```

si LePointeur ne référence pas un Numérique.

APPELER 4D

APPELER 4D
Ne requiert pas de paramètre

Description

APPELER 4D est destinée uniquement à une utilisation avec le compilateur. En effet, seul le moteur de 4D peut détecter un événement. Il était donc nécessaire, dans le cadre d'une base compilée, qu'une routine puisse interroger le moteur de 4D afin de savoir si un événement s'est produit. Cette commande doit donc être utilisée lorsque vous employez la commande **APPELER SUR EVENEMENT**.
Par exemple, si une méthode exécute une boucle dans laquelle aucune commande 4D n'est appelée, la boucle ne pourra pas être interrompue par un process installé à l'aide d' **APPELER SUR EVENEMENT**, et l'utilisateur ne pourra pas ouvrir une autre application. Dans ce cas, **APPELER 4D** doit être insérée pour que 4D puisse intercepter les événements. Bien entendu, n'utilisez pas **APPELER 4D** si vous ne voulez aucune interruption.

Exemple

Dans l'exemple suivant, la boucle ne se terminerait jamais dans une base compilée sans l'aide de **APPELER 4D** :

```
  ` Méthode Traitement quelconque
APPELER SUR EVENEMENT("METHODE EVENEMENT")
◊vbArrêt:=Faux
MESSAGE("Traitement..."+Caractere(13)+"Tapez une touche pour interrompre l'exécution...")
Repeter
  ` Effectuer un traitement sans appel à une commande 4D
   APPELER 4D
Jusque(◊vbArrêt)
APPELER SUR EVENEMENT("")
```

La méthode ***METHODE EVENEMENT*** :

```
  ` Méthode METHODE EVENEMENT
Si(Indefinie(Keycode))
   Keycode:=0
Fin de si
Si(Keycode#0)
   CONFIRMER("Voulez-vous vraiment interrompre cette opération ?")
   Si(OK=1)
      ◊vbArrêt:=Vrai
   Fin de si
Fin de si
```

C_BLOB

C_BLOB ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇒	Nom de méthode
variable	Variable	⇒	Nom(s) de variable(s) ou paramètre(s) ${...} à déclarer

Description

C_BLOB assigne le type BLOB à toutes les variables spécifiées.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_BLOB(${...})** vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_BLOB(${5})** indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande **Nombre de paramètres**.

Exemples

Reportez-vous aux exemples de la section **Commandes du thème Compilateur**.

C_BOOLEEN

C_BOOLEEN ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇨	Nom de méthode
variable	Variable	⇨	Nom(s) de variable(s) à déclarer

Description

C_BOOLEEN affecte le type Booléen à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_BOOLEEN(${...})** vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_BOOLEEN(${5})** indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande **Nombre de paramètres**.

Exemples

Reportez-vous à la section **Commandes du thème Compilateur**.

C_DATE

C_DATE ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇒	Nom de méthode
variable	Variable	⇒	Nom(s) de(s) variable(s) à déclarer

Description

C_DATE affecte le type Date à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_DATE(${...})** vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_DATE(${5})** indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande **Nombre de paramètres**.

Exemples

Reportez-vous à la section **Commandes du thème Compilateur**.

C_ENTIER LONG

C_ENTIER LONG ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇒	Nom de méthode
variable	Variable	⇒	Nom(s) de(s) variable(s) à déclarer

Description

C_ENTIER LONG affecte le type Entier long à chaque *variable* spécifiée.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_ENTIER LONG(${...})** vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_ENTIER LONG(${5})** indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande **Nombre de paramètres**.

Exemples

Reportez-vous à la section **Commandes du thème Compilateur**.

C_HEURE

C_HEURE ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇒	Nom de méthode
variable	Variable	⇒	Nom(s) de(s) variable(s) à déclarer

Description

C_HEURE affecte le type Heure à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_HEURE(${...})** vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_HEURE(${5})** indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande **Nombre de paramètres**.

Exemples

Reportez-vous à la section **Commandes du thème Compilateur**.

C_IMAGE

C_IMAGE ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	➡	Nom de méthode
variable	Variable	➡	Nom(s) de(s) variable(s) à déclarer

Description

C_IMAGE affecte le type Image à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_IMAGE(${...})** vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_IMAGE(${5})** indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande **Nombre de paramètres**.

Exemples

Reportez-vous à la section **Commandes du thème Compilateur**.

C_OBJET

C_OBJET ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇒	Nom de méthode
variable	Variable	⇒	Nom(s) de variable(s) ou paramètre(s) ${...} à déclarer

Description

La commande **C_OBJET** assigne le type *Objet* à toutes les variables spécifiées.

Le type *Objet* est pris en charge par le langage de 4D à compter de la v14. Les objets de ce type sont gérés par les commandes du thème **Objets (Langage)**.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale. Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation

ATTENTION : Cette deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_OBJET**(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_OBJET**(${5}) indique au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type.

Exemple

Reportez-vous à la section **Commandes du thème Compilateur**.

C_POINTEUR

C_POINTEUR ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇒	Nom de méthode
variable	Variable	⇒	Nom(s) de(s) variable(s) à déclarer

Description

C_POINTEUR affecte le type Pointeur à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_POINTEUR(${...})** vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_POINTEUR(${5})** indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande **Nombre de paramètres**.

Exemples

Reportez-vous à la section **Commandes du thème Compilateur**.

C_REEL

C_REEL ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇒	Nom de méthode
variable	Variable	⇒	Nom(s) de(s) variable(s) à déclarer

Description

C_REEL affecte le type Réel (numérique) à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_REEL(${...})** vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_REEL(${5})** indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande **Nombre de paramètres**.

Exemples

Reportez-vous à la section **Commandes du thème Compilateur**.

C_TEXTE

C_TEXTE ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇒	Nom de méthode
variable	Variable	⇒	Nom(s) de(s) variable(s) à déclarer

Description

C_TEXTE affecte le type Texte à chaque *variable* spécifiée.

La première syntaxe de la commande (si le paramètre *méthode* n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre *méthode* est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe **C_TEXTE(${...})** vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration **C_TEXTE(${5})** indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande **Nombre de paramètres**.

Exemples

Reportez-vous à la section **Commandes du thème Compilateur**.

_o_C_ALPHA

_o_C_ALPHA ({méthode ;} taille ; variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	⇒	Nom de méthode
taille	Entier long	⇒	Taille de la chaîne
variable	Variable	⇒	Nom(s) de variable(s) à déclarer

Note de compatibilité

Le fonctionnement de la commande **_o_C_ALPHA** est rigoureusement identique à la celui de commande **C_TEXTE** (le paramètre *taille* est ignoré). Il est désormais conseillé d'utiliser exclusivement **C_TEXTE** dans vos développements 4D.

_o_C_ENTIER

_o_C_ENTIER ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Méthode	➡	Nom de méthode
variable		➡	Nom(s) de(s) variable(s) à déclarer

Note préliminaire

La commande **_o_C_ENTIER** est conservée pour des raisons de compatibilité avec les anciennes bases de données. En effet, en interne 4D et le compilateur retypent les Entiers en Entiers longs. Par exemple :

```
_o_C_ENTIER($MaVar)
$Letype:=Type($MaVar) //$Letype vaut 9 (Est un entier long)
```

_o_C_GRAPHE

_o_C_GRAPHE ({méthode ;} variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
méthode	Chaîne	⇒	Nom de méthode
variable	Variable	⇒	Nom(s) de(s) variable(s) à déclarer

Note de compatibilité

Les variables de type Zone de graphe sont obsolètes et ne sont plus prises en charge depuis 4D v14. Vous devez utiliser des variables images (cf. **GRAPHE**).

Conteneur de données

- Gestion du conteneur de données
- AJOUTER DONNEES AU CONTENEUR
- EFFACER CONTENEUR
- FIXER FICHIER DANS CONTENEUR
- FIXER IMAGE DANS CONTENEUR
- FIXER TEXTE DANS CONTENEUR
- LIRE DONNEES CONTENEUR
- Lire fichier dans conteneur
- LIRE IMAGE DANS CONTENEUR
- Lire texte dans conteneur
- LIRE TYPE DONNEES DANS CONTENEUR
- Tester conteneur

Gestion du conteneur de données

Les commandes du thème “Conteneur de données” permettent de prendre en charge à la fois les actions liées au copier-coller (gestion du Presse-papiers) ainsi que celles liées au glisser-déposer inter-applications.

4D exploite deux conteneurs de données : l’un pour les données copiées (ou coupées), qui est en fait celui du Presse-papiers, déjà présent dans les versions précédentes de 4D et l’autre pour les données en cours de glisser-déposer.
Ces deux conteneurs sont gérés à l’aide des mêmes commandes. Vous accédez à l’un ou à l’autre en fonction du contexte :

- Le conteneur de glisser-déposer est accessible uniquement dans le cadre des événements formulaire Sur début glisser, Sur glisser ou Sur déposer et dans la **Méthode base Sur déposer**. En-dehors de ces contextes, le conteneur de glisser-déposer n’est pas disponible.
- Le conteneur de copier-coller est accessible dans tous les autres cas. A la différence du conteneur de glisser-déposer, il conserve durant la session les données qui y ont été placées, tant qu’il n’a pas été effacé ou réutilisé.

Types de données

Lors du glisser-déposer, différents types de données peuvent être placés et lus dans le conteneur de données. Vous pouvez accéder à un type de données de plusieurs manières :

- via sa signature 4D : la signature 4D est une chaîne de caractères indiquant un type de données référencé par 4D. L’emploi de signatures 4D facilite le développement d’application multi plates-formes, car ces signatures sont identiques sous Mac OS et Windows. Vous trouverez ci-dessous la liste des signatures 4D.
- via un UTI (Uniform Type Identifier, Mac OS uniquement) : la norme UTI, définie par Apple, associe une chaîne de caractères à chaque type d’objet natif. Par exemple, les images GIF ont le type UTI “com.apple.gif”. Les types UTI sont publiés dans les documentations Apple ainsi que par les éditeurs concernés.
- via son numéro ou son nom de format (Windows uniquement) : sous Windows, chaque type de donnée natif est référencé par un numéro (“3”, “12”, etc.) et un nom (“Rich Text Edit”). Par défaut, Microsoft définit plusieurs types natifs appelés formats de données standard. En outre, tout éditeur tiers peut “enregistrer” des noms de formats auprès du système, qui leur attribue un numéro en retour. Pour plus d’informations sur ce principe et sur les types natifs, reportez-vous à la documentation développeur de Microsoft (en particulier http://msdn2.microsoft.com/en-us/library/ms649013.aspx).

Note : Dans les commandes de 4D, les numéros de formats Windows sont manipulés sous forme de textes.

Toutes les commandes du thème “Conteneur de données” peuvent travailler avec chacun de ces types de données. Vous pouvez connaître les types de données présents dans le conteneur dans chacun de ces formats à l’aide de la commande **LIRE TYPE DONNEES DANS CONTENEUR**.

Note : Les types sur 4 caractères (TEXT, PICT ou types personnalisés) sont conservés par compatibilité avec les versions précédentes de 4D.

Signatures 4D

Voici la liste des signatures 4D standard ainsi que leur description :

Signature	Description
"com.4d.private.text.native"	Texte en jeu de caractères natif
"com.4d.private.text.utf16"	Texte en jeu de caractères unicode
"com.4d.private.text.rtf"	Texte enrichi
"com.4d.private.picture.pict"	Image format PICT
"com.4d.private.picture.png"	Image format PNG
"com.4d.private.picture.gif"	Image format GIF
"com.4d.private.picture.jfif"	Image format JPEG
"com.4d.private.picture.emf"	Image format EMF
"com.4d.private.picture.bitmap"	Image format BITMAP
"com.4d.private.picture.tiff"	Image format TIFF
"com.4d.private.picture.pdf"	Document PDF
"com.4d.private.file.url"	Chemin d’accès de fichier

AJOUTER DONNEES AU CONTENEUR

AJOUTER DONNEES AU CONTENEUR (typeDonnées ; données)

Paramètre	Type		Description
typeDonnées	Chaîne	⇒	Type des données à ajouter
données	BLOB	⇒	Données à ajouter au conteneur

Description

AJOUTER DONNEES AU CONTENEUR ajoute dans le conteneur les données du type spécifié dans *typeDonnées* présentes dans le BLOB *données*.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Passez dans *typeDonnées* une valeur définissant le type de données à ajouter. Vous pouvez passer une signature 4D, un type UTI (Mac OS), un nom/numéro de format (Windows), ou un type de 4 caractères (compatibilité). Pour plus d'informations sur ces types, reportez-vous à la section **Gestion du conteneur de données**.

Note pour les utilisateurs Windows : Lorsque la commande est utilisée avec des données de type texte (*typeDonnées* vaut "TEXT", com.4d.private.text.native ou com.4d.private.text.utf16), la chaîne contenue dans le paramètre BLOB *données* doit se terminer par le caractère NULL sous Windows.

Généralement, vous utilisez la commande **AJOUTER DONNEES AU CONTENEUR** pour placer plusieurs instances des mêmes données dans le conteneur de données ou pour y ajouter des valeurs qui ne sont pas du texte ou une image. Pour ajouter de nouvelles données au conteneur, il faut d'abord l'effacer à l'aide de la commande **EFFACER CONTENEUR**.

Si vous voulez effacer le conteneur et y ajouter :

- du texte, utilisez la commande **FIXER TEXTE DANS CONTENEUR**,
- une image, utilisez la commande **FIXER IMAGE DANS CONTENEUR**,
- un chemin d'accès de fichier (glisser-déposer), utilisez la commande **FIXER FICHIER DANS CONTENEUR**.

Notez cependant que si un BLOB contient du texte ou une image, vous pouvez utiliser la commande **AJOUTER DONNEES AU CONTENEUR** pour y ajouter du texte ou une image.

Exemple

A l'aide des commandes du thème Conteneur de données et des BLOBs, vous pouvez écrire des méthodes de Couper/Copier/Coller pour gérer des données structurées au lieu d'une seule information. Dans l'exemple suivant, les deux méthodes projet ***écrire enregistrement dans Presse papiers*** et ***lire enregistrement dans Presse papiers*** vous permettent de traiter un enregistrement comme une information à copier dans le Presse-papiers.

```
   ` Méthode projet écrire enregistrement dans Presse papiers
   ` écrire enregistrement dans Presse papiers ( Numérique )
   ` écrire enregistrement dans Presse papiers ( Numéro de table )

C_ENTIER LONG($1;$vlChamp;$vlTypeChamp)
C_POINTEUR($vpTable;$vpChamp)
C_ALPHA(255;$vaNomDoc)
C_TEXTE($vtDonnéesEnregistrement;$vtDonnéesChamp)
C_BLOB($vxDonnéesEnregistrement)

   ` Effacer le Presse-papiers (il restera vide s'il n'y a pas d'enregistrement courant)
EFFACER CONTENEUR
   ` Obtenir un pointeur vers la table dont le numéro est passé en paramètre
$vpTable:=Table($1)
   ` S'il y a un enregistrement courant pour cette table
Si((Numero enregistrement($vpTable->)>=0) | (Nouvel enregistrement($vpTable->)))
   ` Initialiser la variable Texte qui contiendra l'image de texte de l'enregistrement
   $vtDonnéesEnregistrement:=""
   ` Pour chaque champ de l'enregistrement :
   Boucle($vlChamp;1;Lire numero dernier champ($1))
   ` Obtenir le type du champ
      LIRE PROPRIETES CHAMP($1;$vlChamp;$vlTypeChamp)
   ` Obtenir un pointeur vers le champ
      $vpChamp:=Champ($1;$vlChamp)
   ` Selon le type du champ, copier (ou non) ses données de façon appropriée
      Au cas ou
         :(($vlTypeChamp=Est un champ alpha) | ($vlTypeChamp=Est un texte))
            $vtDonnéesChamp:=$vpChamp->
         :(($vlTypeChamp=Est un numérique) | ($vlTypeChamp=Est un entier) | ($vlTypeChamp=Est un entier
long)| ($vlTypeChamp=Est une date)|($vlTypeChamp=Est une heure))
            $vtDonnéesChamp:=Chaine($vpChamp->)
         :($vlTypeChamp=Est un booléen)
            $vtDonnéesChamp:=Chaine(Num($vpChamp->);"Oui;;Non")
         Sinon
   ` Passer et ignorer les autres types de champs
            $vtDonnéesChamp:=""
      Fin de cas
   ` Accumuler les données sur le champ dans une variable texte qui stocke l'image de texte de l'enregistrement
      $vtDonnéesEnregistrement:=$vtDonnéesEnregistrement+Nom du
```

```
champ($1;$vlChamp)+":"+Caractere(9)+$vtDonnéesChamp+CR
   ` Note : La méthode CR retourne Caractere(13) sous Mac OS et Caractere(13)+Caractere(10) sous Windows
   Fin de boucle
  ` Mettre l'image de texte de l'enregistrement dans le Presse-papiers
   FIXER TEXTE DANS CONTENEUR($vtDonnéesEnregistrement)
  ` Nommez le fichier d'Album dans le Dossier temporaire
   $vaNomDoc:=Dossier temporaire+"Album"+Chaine(1+(Hasard%99))
  ` Supprimer le fichier d'Album s'il existe (il faut tester une erreur ici)
   SUPPRIMER DOCUMENT($vaNomDoc)
  ` Créez le fichier d'Album
   REGLER SERIE(10;$vaNomDoc)
  ` Envoyer l'enregistrement entier dans le Presse-papiers
   ENVOYER ENREGISTREMENT($vpTable->)
  ` Fermer le fichier d'Album
   REGLER SERIE(11)
  ` Charger le fichier d'Album dans un BLOB
   DOCUMENT VERS BLOB($vaNomDoc;$vxDonnéesEnregistrement)
  ` Nous n'avons plus besoin du fichier d'Album
   SUPPRIMER DOCUMENT($vaNomDoc)
  ` Ajouter l'image complète de l'enregistrement dans le Presse-papiers
  ` Note: nous utilisons le type de données "4Drc" de façon arbitraire
   AJOUTER DONNEES AU CONTENEUR("4Drc";$vxDonnéesEnregistrement)
  ` Le Presse-papiers contient :
  ` (1) Une image de texte de l'enregistrement (comme illustré dans les copies d'écran ci-dessous)
  ` (2) Une image entière de l'enregistrement (y compris les images, sous-tables et les champs de type BLOB)
Fin de si
```

Lors de la saisie d'un enregistrement, si vous appliquez la méthode ***écrire enregistrement dans Presse papiers*** à la table, le Presse-papiers contiendra le texte de l'enregistrement et également l'image entière de l'enregistrement.

Vous pouvez coller cette image de l'enregistrement dans un autre enregistrement, à l'aide de la méthode ***lire enregistrement dans Presse papiers***, qui est la suivante :

```
  ` Méthode lire enregistrement dans Presse papiers
  ` lire enregistrement dans Presse papiers ( Numéro )
  ` lire enregistrement dans Presse papiers ( Numéro de table )
C_ENTIER LONG($1;$vlChamp;$vlTypeChamp;$vlPosCR;$vlPosColon)
C_POINTEUR($vpTable;$vpChamp)
C_ALPHA(255;$vaNomDoc)
C_BLOB($vxDonnéeesPressePapiers)
C_TEXTE($vtDonnéeesPressePapiers;$vtDonnéesChamp)

  ` Obtenir un pointeur vers la table dont le numéro est passé en tant que paramètre
$vpTable:=Table($1)
  ` S'il y a un enregistrement courant pour cette table
Si((Numero enregistrement($vpTable->)>=0) | (Nouvel enregistrement($vpTable->)))
   Au cas ou
  ` Est-ce que le Presse-papiers contient une image entière de l'enregistrement ?
      :(Tester conteneur("4Drc")>0)
  ` Si oui, extraire le contenu du Presse-papiers
         LIRE DONNEES CONTENEUR("4Drc";$vxDonnéesPressePapiers)
  ` Nommer le fichier d'Album dans le Dossier temporaire
         $vaNomDoc:=Dossier temporaire+"Album"+Chaine(1+(Hasard%99))
  ` Supprimer le fichier d'Album s'il existe (il faut tester l'erreur ici)
         SUPPRIMER DOCUMENT($vaNomDoc)
  ` Enregistrer le BLOB dans le fichier d'Album
         BLOB VERS DOCUMENT($vaNomDoc;$vxDonnéesPressePapiers)
  ` Ouvrir le fichier d'Album
         REGLER SERIE(10;$vaNomDoc)
  ` Recevoir l'enregistrement entier du fichier d'Album
         RECEVOIR ENREGISTREMENT($vpTable->)
  ` Fermer le fichier d'Album
         REGLER SERIE(11)
  ` Nous n'avons plus besoin du fichier d'Album
         SUPPRIMER DOCUMENT($vaNomDoc)
  ` Est-ce que le Presse-papiers contient du texte ?
      :(Tester conteneur("TEXT")>0)
  ` Extraire le texte du Presse-papiers
         $vtDonnéesPressePapiers:=Lire texte dans conteneur
  ` Initialiser le numéro de champ à incrémenter
         $vlChamp:=0
         Repeter
  ` Chercher la ligne de champ suivante dans le texte
            $vlPosCR:=Position(CR;$vtDonnéesPressePapiers)
            Si($vlPosCR>0)
  ` Extraire la ligne de champ
               $vtDonnéesChamp:=Sous chaine($vtDonnéesPressePapiers;1;$vlPosCR-1)
  ` S'il y a un signe deux points ":"
               $vlPosColon:=Position(":";$vtDonnéesChamp)
               Si($vlPosColon>0)
  ` Récupérer seulement les données de champ (supprimer le nom du champ)
```

```
                  $vtDonnéesChamp:=Sous chaine($vtDonnéesChamp;$vlPosColon+2)
               Fin de si
   ` Incrémenter le numéro du champ
               $vlChamp:=$vlChamp+1
   ` Le Presse-papiers peut contenir plus de données dont nous n'avons pas besoin...
               Si($vlChamp<=Lire numero dernier champ($vpTable))
   ` Obtenir le type du champ
                  LIRE PROPRIETES CHAMP($1;$vlChamp;$vlTypeChamp)
   ` Obtenir un pointeur vers le champ
                  $vpChamp:=Champ($1;$vlChamp)
   ` Selon le type du champ, copier (ou non) le texte d'une manière appropriée
                  Au cas ou
                     :(($vlTypeChamp=Est un champ alpha) | ($vlTypeChamp=Est un texte))
                        $vpChamp->:=$vtDonnéesChamp
                     :(($vlTypeChamp=Est un numérique) | ($vlTypeChamp=Est un entier) | ($vlTypeChamp=Est un
entier long))
                        $vpChamp->:=Num($vtDonnéesChamp)
                     :($vlTypeChamp=Est une date)
                        $vpChamp->:=Date($vtDonnéesChamp)
                     :($vlTypeChamp=Est une heure)
                        $vpChamp->:=Heure($vtDonnéesChamp)
                     :($vlTypeChamp=Est un booléen)
                        $vpChamp->:=($vtDonnéesChamp="Oui")
                     Sinon
   ` Passer et ignorer les autres types de données
                  Fin de cas
               Sinon
   ` Tous les champs ont été affectés, sortir de la boucle
                  $vtDonnéesPressePapiers:=""
               Fin de si
   ` Eliminer le texte qui vient d'être extrait
               $vtDonnéeesPressePapiers:=Sous chaine($vtDonnéeesPressePapiers;$vlPosCR+Longueur(CR))
            Sinon
   ` Aucun délimiteur trouvé, sortir de la boucle
               $vtDonnéesPressePapiers:=""
            Fin de si
   ` Répéter jusqu'à ce que nous ayons des données
         Jusque(Longueur($vtDonnéesPressePapiers)=0)
      Sinon
         ALERTE("Le Presse-papiers ne contient pas de données pouvant être collées en tant qu'enregistrement.")
   Fin de cas
Fin de si
```

Variables et ensembles système

Si les données dans le BLOB sont correctement ajoutées au conteneur, la variable système OK prend la valeur 1. Sinon, OK est mise à 0 et une erreur peut être générée.

EFFACER CONTENEUR

EFFACER CONTENEUR
Ne requiert pas de paramètre

Description

EFFACER CONTENEUR efface entièrement le conteneur de données. Si le conteneur contient plusieurs instances des mêmes données, toutes les instances sont effacées. Après un appel à **EFFACER CONTENEUR**, le conteneur de données est vide.

Vous devez appeler **EFFACER CONTENEUR** une fois avant de placer des nouvelles données dans le conteneur à l'aide de la commande **AJOUTER DONNEES AU CONTENEUR**, car cette dernière n'efface pas le conteneur avant d'y coller des données.

Si vous appelez **EFFACER CONTENEUR** une fois et puis appelez **AJOUTER DONNEES AU CONTENEUR** plusieurs fois, vous pouvez couper ou copier les mêmes données sous des formats différents.

En revanche, les commandes **FIXER TEXTE DANS CONTENEUR** et **FIXER IMAGE DANS CONTENEUR** effacent automatiquement le conteneur avant d'y placer des données.

Exemple 1

Le code suivant efface le conteneur puis y ajoute des données :

```
EFFACER CONTENEUR ` Effacer le conteneur
AJOUTER DONNEES AU CONTENEUR("com.4d.private.picture.gif";$vxSomeData) ` Ajouter des images gif
AJOUTER DONNEES AU CONTENEUR("com.4d.private.text.rtf";$vxSylkData) ` Ajouter du texte RTF
```

Exemple 2

Reportez-vous à l'exemple de la commande **AJOUTER DONNEES AU CONTENEUR**.

FIXER FICHIER DANS CONTENEUR

FIXER FICHIER DANS CONTENEUR (fichier {; *})

Paramètre	Type		Description
fichier	Chaîne	⇒	Nom de fichier ou Chemin d'accès complet de fichier
*	Opérateur	⇒	Si passé = ajouter, Si omis = remplacer

Description

La commande **FIXER FICHIER DANS CONTENEUR** ajoute dans le conteneur de données le chemin d'accès complet du fichier passé dans le paramètre *fichier*. Cette commande permet de mettre en place des interfaces autorisant le glisser-déposer d'objets 4D vers des fichiers sur le bureau par exemple.

Vous pouvez passer dans *fichier* soit un chemin d'accès complet, soit un nom de fichier simple (sans chemin d'accès). Dans ce dernier cas, le fichier doit être situé à côté du fichier de structure de la base.

La commande admet l'étoile * en paramètre optionnel. Par défaut, lorsque ce paramètre est omis, la commande remplace le contenu du conteneur de données par le dernier chemin d'accès défini par *fichier*. Si vous passez ce paramètre, la commande ajoute le *fichier* au conteneur de données. Il peut ainsi contenir une "pile" de chemins d'accès de fichiers. Dans les deux cas, si des données autres que des chemins d'accès étaient présentes dans le conteneur, elles sont effacées.

Note : Le conteneur de données est en lecture seule pendant l'événement formulaire Sur glisser. Il n'est pas possible d'utiliser cette commande dans ce contexte.

FIXER IMAGE DANS CONTENEUR

FIXER IMAGE DANS CONTENEUR (image)

Paramètre	Type		Description
image	Image	⇨	Image à placer dans le conteneur de données

Description

FIXER IMAGE DANS CONTENEUR place dans le conteneur de données une copie de l'image que vous avez passée dans *image*. Les données éventuellement présentes dans le conteneur sont préalablement effacées.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

L'image est transportée dans son format natif (jpeg, tif, png, etc.).

Après avoir placé l'image dans le conteneur, vous pouvez la récupérer à l'aide de la commande **LIRE IMAGE DANS CONTENEUR** ou par exemple **LIRE DONNEES CONTENEUR("com.4d.private.picture.gif";...)**.

Exemple

Dans une fenêtre flottante, vous affichez un formulaire contenant le tableau *tabNomEmployés* qui liste les noms des employés stockés dans la table [Employés]. Chaque fois que vous cliquez sur un nom, vous voulez copier la photographie de l'employé dans le Presse-papiers. Dans la méthode objet du tableau, vous écrivez :

```
Si(tabNomEmployés#0)
   CHERCHER([Employés];[Employés]Nom=tabNomEmployés{tabNomEmployés})
   Si(Taille image([Employés]Photo)>0)
      FIXER IMAGE DANS CONTENEUR([Employés]Photo) ` Copier la photo de l'employée
   Sinon
      EFFACER CONTENEUR ` Aucune photo trouvée ou aucun enregistrement trouvé
   Fin de si
Fin de si
```

Variables et ensembles système

Si une copie de l'image est correctement collée dans le conteneur, la variable système OK prend la valeur 1. S'il n'y a pas assez de mémoire pour coller l'image dans le Presse-papiers, la variable système OK prend la valeur 0, mais aucune erreur n'est générée.

FIXER TEXTE DANS CONTENEUR

FIXER TEXTE DANS CONTENEUR (texte)

Paramètre	Type		Description
texte	Chaîne	⇨	Texte à placer dans le conteneur de données

Description

FIXER TEXTE DANS CONTENEUR place une copie du texte que vous avez passé dans *texte* dans le conteneur de données. Les données éventuellement présentes dans le conteneur sont auparavant effacées.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Vous pouvez récupérer le texte collé dans le conteneur de données à l'aide de la fonction **Lire texte dans conteneur** ou en appelant par exemple **LIRE DONNEES CONTENEUR("com.4d.private.text.native";...)**.

Les expressions de type Texte de 4D peuvent contenir jusqu'à 2 Go de texte.

Note : Le conteneur de données est en lecture seule pendant l'événement formulaire Sur glisser. Il n'est pas possible d'utiliser cette commande dans ce contexte.

Exemple

Référez-vous à l'exemple de la commande **AJOUTER DONNEES AU CONTENEUR**.

Variables et ensembles système

Si la copie du texte est correctement placée dans le conteneur de données, la variable système OK prend la valeur 1. S'il n'y a pas assez de mémoire pour placer une copie du texte dans le conteneur, la variable système OK prend la valeur 0, mais aucune erreur n'est générée.

LIRE DONNEES CONTENEUR

LIRE DONNEES CONTENEUR (typeDonnées ; données)

Paramètre	Type		Description
typeDonnées	Chaîne	➡	Type de données à extraire du conteneur
données	BLOB	⬅	Données extraites du conteneur

Description

LIRE DONNEES CONTENEUR retourne dans le champ ou la variable de type BLOB *données* les données qui se trouvent dans le conteneur de données et dont le type est passé dans *typeDonnées*. (Si le conteneur de données contient du texte copié depuis 4D, le jeu de caractères du BLOB sera probablement UTF-16.)

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Passez dans *typeDonnées* une valeur définissant le type de données à extraire. Vous pouvez passer une signature 4D, un type UTI (Mac OS), un nom/numéro de format (Windows), ou un type de 4 caractères (compatibilité). Pour plus d'informations sur ces types, reportez-vous à la section **Gestion du conteneur de données**.

Note : Il n'est pas possible de lire les données de type fichier avec cette commande, pour cela vous devez utiliser la commande **Lire fichier dans conteneur**.

Exemple

Les méthodes objet suivantes sont celles de deux boutons qui copient et collent des données dans le tableau *taOptions* (pop up menu, liste déroulante...) se trouvant dans le formulaire :

```
 ` Méthode objet bCopiertaOptions
Si(Taille tableau(taOptions)>0) ` Est-ce qu'il y a quelque chose à copier ?
   VARIABLE VERS BLOB(taOptions;$vxClipData) ` Mettre les éléments du tableau dans un BLOB
   EFFACER CONTENEUR ` Vider le Presse-papiers
   AJOUTER DONNEES AU CONTENEUR("artx";taOptions) ` Le type de données est choisi arbitrairement
Fin de si

 ` Méthode objet bCollertaOptions
Si(Tester conteneur("artx")>0) ` Est-ce qu'il y a les données du type "artx" dans le Presse-papiers?
   LIRE DONNEES CONTENEUR("artx";$vxClipData) ` Extraire les données du Presse-papiers
   BLOB VERS VARIABLE($vxClipData;taOptions) ` Remplir le tableau avec les données venant du BLOB
   taOptions:=0 ` Réinitialiser l'élément sélectionné du tableau
Fin de si
```

Variables et ensembles système

Si les données sont extraites correctement, la variable OK prend la valeur 1. Sinon, elle prend la valeur 0.

Lire fichier dans conteneur

Lire fichier dans conteneur (indiceN) -> Résultat

Paramètre	Type		Description
indiceN	Entier long	→	Nième fichier inclus dans le glisser
Résultat	Chaîne	←	Chemin d'accès de fichier extrait du conteneur de données

Description

La commande **Lire fichier dans conteneur** retourne le chemin d'accès absolu d'un fichier inclus dans une opération de glisser-déposer. Plusieurs fichiers pouvant être sélectionnés et déplacés simulanément, le paramètre *indiceN* permet de désigner un fichier parmi l'ensemble des fichiers sélectionnés.
S'il n'y a pas de Nième fichier dans le conteneur de données, la commande retourne une chaîne vide.

Exemple

L'exemple suivant permet de récupérer dans un tableau tous les chemins d'accès des fichiers inclus dans le glisser-déposer :

```
TABLEAU TEXTE($tabFichiers;0)
C_TEXTE($vtfichier)
C_ENTIER($n)
$n:=1
Repeter
   $vtfichier:=Lire fichier dans conteneur($n)
   Si($vtfichier#"")
      AJOUTER A TABLEAU($tabFichiers;$vtfichier)
      $n:=$n+1
   Fin de si
Jusque($vtfichier="")
```

LIRE IMAGE DANS CONTENEUR

LIRE IMAGE DANS CONTENEUR (image)

Paramètre	Type		Description
image	Image	⇦	Image extraite du conteneur de données

Description

LIRE IMAGE DANS CONTENEUR retourne l'image présente dans le conteneur de données dans le champ ou la variable *image*.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Exemple

Ci-dessous, la méthode objet d'un bouton affecte l'image au format jpeg ou gif présente dans le conteneur de données, s'il y en a une, au champ [Employés]Photo :

```
Si((Tester conteneur("com.4d.private.picture.jfif")>0)|(Tester conteneur("com.4d.private.picture.gif")>0))
   LIRE IMAGE DANS CONTENEUR([Employés]Photo)
Sinon
   ALERTE("Le Presse-papiers ne contient pas d'image.")
Fin de si
```

Variables et ensembles système

Si l'image est correctement extraite, OK prend la valeur 1. Sinon, OK prend la valeur 0.

Lire texte dans conteneur

Lire texte dans conteneur -> Résultat

Paramètre	Type		Description
Résultat	Chaîne		Texte présent dans le conteneur de données

Description

Lire texte dans conteneur retourne le texte présent dans le conteneur de données.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Si le conteneur de données contient du texte enrichi (par exemple au format RTF), le texte conserve ses attributs au moment du déposer ou du coller, si la zone de destination est compatible.

A noter que les champs et variables de type Texte de 4D peuvent contenir jusqu'à 2 Go de texte.

Variables et ensembles système

Si le texte est correctement extrait, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

LIRE TYPE DONNEES DANS CONTENEUR

LIRE TYPE DONNEES DANS CONTENEUR (signatures4D ; typesNatifs {; nomsFormats})

Paramètre	Type		Description
signatures4D	Tableau texte	⇐	Signatures 4D des types de données
typesNatifs	Tableau texte	⇐	Types de données natifs
nomsFormats	Tableau texte	⇐	Noms des formats (Windows uniquement), chaînes vides sous Mac OS

Description

La commande **LIRE TYPE DONNEES DANS CONTENEUR** permet d'obtenir la liste des types de données présents dans le conteneur. Cette commande doit généralement être utilisée dans le contexte d'un glisser-déposer, dans le cadre des événements formulaire Sur déposer ou Sur glisser de l'objet de destination. Elle permet notamment de vérifier la présence d'un type de données spécifique dans le conteneur.

Cette commande retourne les types de données sous plusieurs formes différentes via deux (ou trois) tableaux :

- le tableau *signatures4D* contient les types de données exprimés à l'aide de leur signature 4D interne (par exemple "com.4d.private.picture.gif"). Si un type de données présent n'est pas reconnu par 4D, une chaîne vide ("") est retournée dans le tableau.
- le tableau *typesNatifs* contient les types de données exprimés à l'aide de leur type natif. Le format des types natifs diffère entre Mac OS et Windows :
  - Sous Mac OS, les types natifs sont exprimés sous forme d'UTI (UniformType Identifier).
  - Sous Windows, les types natifs sont exprimés sous forme de numéros, chaque numéro étant associé à un nom de format. Le tableau *typesNatifs* contient ces numéros sous forme de chaîne ("3", "12", etc.). Si vous souhaitez utiliser des libellés plus explicites, il est recommandé d'utiliser le tableau facultatif *nomsFormats*, qui contient le nom de format des types natifs sous Windows.

  Le tableau *typesNatifs* permet de prendre en charge tout type de données présent dans le conteneur, y compris des données dont le type n'est pas référencé par 4D.
- Sous Windows, vous pouvez également passer le tableau *nomsFormats*, qui reçoit les noms des types de données présents dans le conteneur. Les valeurs retournées dans ce tableau peuvent être utilisées par exemple pour construire un pop up menu de sélection de format. Sous Mac OS, le tableau *nomsFormats* retourne des chaînes vides.

Pour plus d'informations sur les types de données pris en charge, reportez-vous à la section **Gestion du conteneur de données**.

Tester conteneur

Tester conteneur (typeDonnées) -> Résultat

Paramètre	Type		Description
typeDonnées	Chaîne	⇒	Type de données
Résultat	Entier long	⇐	Taille (en octets) des données présentes dans le conteneur ou code d'erreur

Description

Tester conteneur vous permet de savoir s'il y a des données du type *typeDonnées* dans le conteneur de données.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Si le conteneur de données est vide ou ne contient pas de données du type spécifié, la fonction retourne une erreur -102. Si le conteneur contient des données du type spécifié, la fonction retourne la taille des données exprimée en octets.

Passez dans *typeDonnées* une valeur définissant le type de données à tester. Vous pouvez passer une signature 4D, un type UTI (Mac OS), un nom/numéro de format (Windows), ou un type de 4 caractères (compatibilité). Pour plus d'informations sur ces types, reportez-vous à la section **Gestion du conteneur de données**.

Après avoir vérifié que le conteneur contient bien des données du type que vous voulez, vous pouvez les récupérer à l'aide d'une des commandes suivantes :

- Si le conteneur contient du texte, vous pouvez l'extraire à l'aide de la commande **Lire texte dans conteneur**, qui retourne une valeur texte. Sinon, vous pouvez utiliser la commande **LIRE DONNEES CONTENEUR**, qui retourne le texte dans un BLOB.
- Si le conteneur contient une image, vous pouvez l'extraire à l'aide de la commande **LIRE IMAGE DANS CONTENEUR**, qui retourne l'image dans un champ ou une variable. Sinon, vous pouvez utiliser la commande **LIRE DONNEES CONTENEUR**, qui retourne l'image dans un BLOB.
- Si le conteneur contient un chemin d'accès de fichier, vous pouvez l'extraire à l'aide de la commande **Lire fichier dans conteneur**, qui retourne le chemin d'accès du fichier.
- Pour tout type de données, vous pouvez utiliser la commande **LIRE DONNEES CONTENEUR**, qui retourne les données dans un BLOB.

Exemple 1

L'exemple suivant teste si le Presse-papiers contient une image et, si oui, la copie dans une variable 4D :

```
Si(Tester conteneur(Données image)=1) //Y a-t-il une image dans le Presse-papiers ?
   LIRE IMAGE DANS CONTENEUR($vPicVariable) //Si oui, extraire l'image du Presse-papiers
Sinon
   ALERTE("Il n'y a pas d'image dans le Presse-papiers.")
Fin de si
```

Exemple 2

Généralement, après un couper ou un copier, les applications placent des données de type Texte ou Image dans le Presse-papiers, ces deux types de données standard sont reconnus par la plupart des applications. Cependant, une application peut placer dans le Presse-papiers plusieurs copies des mêmes données sous des formats différents. Par exemple, chaque fois que vous copiez ou coupez un tableau, l'application tableur peut placer les données dans un format propriétaire — par exemple, 'SPSH' — ou dans les formats SYLK et TEXT. La copie 'SPSH' contient les données structurées dans le format interne de l'application. La copie SYLK contient les mêmes données, mais dans le format SYLK, reconnu par la plupart des tableurs. Enfin, la copie TEXT contient les mêmes données, mais sans les informations de formatage supplémentaires présentes dans les formats SYLK ou 'SPSH'. Donc, lorsque vous écrivez des routines de Couper/Copier/Coller entre 4D et une application tableur, en prenant l'hypothèse que vous connaissez la description du format 'SPSH' et que vous pouvez analyser les données SYLK, vous pouvez écrire le code suivant :

```
Au cas ou
  ` D'abord, vérifier si le Presse-papiers contient les données venant du tableur
   :(Tester conteneur('SPSH')>0)
  ` ...
  ` Ensuite, vérifier si le Presse-papiers contient des données au format SYLK
   :(Tester conteneur('SYLK')>0)
  ` ...
  ` Enfin, vérifier si le Presse-papiers contient des données au format TEXT
   :(Tester conteneur('TEXT')>0)
  ` ...
Fin de cas
```

Autrement dit, vous essayez d'extraire du Presse-papiers la copie des données la plus riche en informations originales.

Exemple 3

Vous voulez déplacer des données en format privé entre divers objets de votre formulaire. Vous pouvez écrire :

```
 //objet source
Si(Evenement formulaire=Sur début glisser)
   AJOUTER DONNEES AU CONTENEUR("some.private.data";$data)
Fin de si
```

```
 //objet cible
Si(Evenement formulaire=Sur glisser)
   $0:=Choisir(Tester conteneur("some.private.data")>0;0;-1)
Fin de si
```

Exemple 4

Référez-vous à l'exemple de la commande **AJOUTER DONNEES AU CONTENEUR**.

Correcteur orthographique

- Prise en charge des dictionnaires Hunspell
- SPELL AJOUTER AU DICTIONNAIRE UTILISATEUR
- SPELL CORRECTION ORTHOGRAPHIQUE
- SPELL FIXER DICTIONNAIRE COURANT
- SPELL Lire dictionnaire courant
- SPELL LIRE LISTE DICTIONNAIRES
- SPELL VERIFIER TEXTE

Prise en charge des dictionnaires Hunspell

A compter des versions 12.4 et 13, 4D prend en charge les dictionnaires OpenSource "Hunspell". Pour plus d'informations sur ces dictionnaires, reportez-vous au site http://hunspell.sourceforge.net/

Le format retenu par 4D est celui utilisé par MySpell et OpenOffice 2.x : un fichier .aff et un fichier .dic qui portent le même nom. Par exemple, le dictionnaire "fr-moderne" et composé des fichiers *fr-moderne.aff* et *fr-moderne.dic*.

Pour pouvoir utiliser un dictionnaire Hunspell dans une application 4D, vous devez installer ses fichiers .aff et .dic à l'un des emplacements suivants (premier niveau) :

- dans l'application 4D : <4D>/Resources/Spellcheck/Hunspell/
- dans la base 4D : <Fichiers_Base>/Resources/Hunspell

Les deux emplacements sont compatibles : le dossier de la base est analysé en premier lieu, puis est complété par celui de l'application 4D, ce qui permet d'encapsuler des dictionnaires spécialisés avec vos bases 4D. Si deux dictionnaires du même nom existent aux deux emplacements, c'est celui de la base qui est pris en compte.

Vous pouvez télécharger des dictionnaires Hunspell à l'adresse suivante : http://wiki.services.openoffice.org/wiki/Dictionaries

Les utilisateurs peuvent également enrichir des dictionnaires existants ou en créer de nouveaux. Le principe de création est identique à celui des dictionnaires utilisateurs Cordial (cf. section **Annexe D : Utilisation de dictionnaires spécialisés** du manuel *Mode Développement*). Les dictionnaires utilisateurs sont stockés au format UTF-8.

SPELL AJOUTER AU DICTIONNAIRE UTILISATEUR

SPELL AJOUTER AU DICTIONNAIRE UTILISATEUR (mots)

Paramètre	Type		Description
mots	Texte, Tableau texte	⇨	Mot ou liste de mots à ajouter au dictionnaire utilisateur

Description

La commande **SPELL AJOUTER AU DICTIONNAIRE UTILISATEUR** permet d'ajouter un ou plusieurs mot(s) au dictionnaire utilisateur courant.

Le dictionnaire utilisateur est un dictionnaire contenant les mots ajoutés par l'utilisateur au dictionnaire courant. Ce dictionnaire est un fichier nommé *UserDictionaryxxx.dic* (ou xxx représente l'ID du dictionnaire courant) et automatiquement créé dans le dossier 4D courant. Il existe un dictionnaire utilisateur par dictionnaire courant utilisé.

Vous pouvez passer dans *mots* une chaîne texte ou un tableau texte contenant le ou les mot(s) à ajouter dans le dictionnaire utilisateur. Si un mot est déjà présent dans le dictionnaire, il est ignoré par la commande.

Exemple

Ajout de noms propres au dictionnaire utilisateur :

```
TABLEAU TEXTE($tTwords;0)
AJOUTER A TABLEAU($tTwords;"4D")
AJOUTER A TABLEAU($tTwords;"Wakanda")
AJOUTER A TABLEAU($tTwords;"Clichy")
SPELL AJOUTER AU DICTIONNAIRE UTILISATEUR($tTwords)
```

SPELL CORRECTION ORTHOGRAPHIQUE

SPELL CORRECTION ORTHOGRAPHIQUE
Ne requiert pas de paramètre

Description

La commande **SPELL CORRECTION ORTHOGRAPHIQUE** déclenche la vérification de l'orthographe du champ ou de la variable ayant le focus dans le formulaire affiché à l'écran. L'objet vérifié doit être de type Alpha ou Texte.

Note : Si vous souhaitez déclencher la correction orthographique à partir d'un bouton dans le formulaire, assurez-vous qu'il ne dispose pas de la propriété "Focusable".

La vérification débute par le premier mot du champ ou de la variable. Si un mot inconnu est détecté, la boîte de dialogue de correction apparaît (pour plus d'informations, reportez-vous au manuel Mode Développement de 4D). 4D utilise le dictionnaire courant (correspondant à la langue de l'application) sauf si vous avez utilisé la commande **SPELL FIXER DICTIONNAIRE COURANT**.

Attention : La commande **SPELL CORRECTION ORTHOGRAPHIQUE** agit sur le texte en cours de saisie dans le formulaire, et non sur sa source de données associée (variable ou champ). Ce principe implique que si la commande est appelée depuis les événements formulaire Sur données modifiées ou Sur perte focus (non recommandé), elle n'aura pas d'effet sur le texte stocké car à ce moment, 4D a déjà assigné le texte saisi à la source de données. Dans ce cas, vous devez alors assigner vous-même le résultat modifié à la source de données, via la commande **Lire texte edite**. Par exemple :

```
Si(Evenement formulaire=Sur données modifiées)
   SPELL CORRECTION ORTHOGRAPHIQUE
   theVariable:=Lire texte edite
Fin de si
```

SPELL FIXER DICTIONNAIRE COURANT

SPELL FIXER DICTIONNAIRE COURANT (dictionnaire)

Paramètre	Type		Description
dictionnaire	Entier long, Texte	➡	ID, Nom ou Code de langue du dictionnaire à utiliser pour la correction orthographique

Description

La commande **SPELL FIXER DICTIONNAIRE COURANT** provoque le remplacement du dictionnaire courant par celui spécifié par le paramètre *dictionnaire*. Le dictionnaire courant est utilisé pour la correction orthographique intégrée de 4D (pour plus d'informations, reportez-vous au manuel *Mode Développement*) ainsi qu'à celle des plug-ins 4D Write et 4D View. La modification du dictionnaire courant est immédiatement répercutée dans tous les process de la base pour la session, ainsi que dans les zones des plug-ins 4D Write et 4D View.

Par défaut, 4D utilise :

- sous Windows, le dictionnaire Hunspell correspondant à la langue de l'application,
- sous OS X, le correcteur orthographique natif.

Note : Pour plus d'informations sur les dictionnaires Hunspell, reportez-vous à la section **Prise en charge des dictionnaires Hunspell**.

Vous pouvez changer de dictionnaire à l'aide du paramètre *dictionnaire*. Vous pouvez passer soit :

- un numéro d'ID de dictionnaire Hunspell (retourné par la commande **SPELL LIRE LISTE DICTIONNAIRES**),
- un nom de dictionnaire Hunspell (correspondant au nom du fichier de dictionnaire Hunspell avec ou sans extension),
- un code de langue BCP 47, ISO 639-1 ou ISO 639-2. Par exemple, "fr-FR" désigne le français de France et "fr-BE" le français de Belgique avec le code de langue BCP 47. Ces codes sont redirigés en interne vers le dictionnaire courant correspondant (Hunspell ou natif OS X).

Note de compatibilité : Dans les versions précédentes de 4D, les dictionnaires "Cordial" étaient également pris en charge. Par compatibilité, il reste possible de passer un numéro de dictionnaire "Cordial" dans le paramètre *dictionnaire* (valeur ou constante du thème "**Dictionnaires**"). Dans ce cas toutefois, le dictionnaire est redirigé en interne vers un dictionnaire Hunspell équivalent (ou le dictionnaire natif sous OS X).

Variables et ensembles système

Si le *dictionnaire* est correctement chargé, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est retournée.

Exemple

Chargement du dictionnaire "fr-classique" présent dans le dossier Hunspell :

```
SPELL FIXER DICTIONNAIRE COURANT("fr-classique")
    // SPELL FIXER DICTIONNAIRE COURANT("FR-classique.dic") est valide
```

SPELL Lire dictionnaire courant

SPELL Lire dictionnaire courant -> Résultat

Paramètre	Type	Description
Résultat	Entier long	ID du dictionnaire utilisé pour la correction orthographique

Description

La commande **SPELL Lire dictionnaire courant** retourne le numéro d'ID du dictionnaire en cours d'utilisation.

Exemple

On souhaite afficher la langue du dictionnaire courant :

```
 // Liste des dictionnaires chargés
SPELL LIRE LISTE DICTIONNAIRES($IDs_al;$Codes_at;$Noms_at)
$curLangCode:=SPELL Lire dictionnaire courant
$countryName:=$Noms_at{Chercher dans tableau($IDs_al;$curLangCode)}
 // Affichage du message
ALERTE("Dictionnaire courant : "+$countryName) // Espagnol
```

SPELL LIRE LISTE DICTIONNAIRES

SPELL LIRE LISTE DICTIONNAIRES (langID ; langFichiers ; langNoms)

Paramètre	Type		Description
langID	Tableau entier long	⇦	ID uniques des langues
langFichiers	Tableau texte	⇦	Noms des fichiers de langue installés
langNoms	Tableau texte	⇦	Noms locaux des langues

Description

La commande **SPELL LIRE LISTE DICTIONNAIRES** retourne dans les tableaux *langID*, *langFichiers* et *langNoms* les IDs, les noms de fichiers et les noms des langues correspondant aux fichiers de dictionnaires Hunspell installés sur la machine.

Note : Pour plus d'informations sur les dictionnaires Hunspell, reportez-vous à la section **Prise en charge des dictionnaires Hunspell**.

- *langID* reçoit les numéros d'ID générés automatiquement et utilisables avec la commande **SPELL FIXER DICTIONNAIRE COURANT**.
  A noter que les IDs sont uniques et basés sur les noms de fichiers. Cette commande est donc principalement utile en phase de développement, il n'est pas nécessaire de regénérer des IDs à chaque exécution de la base.
- *langFichiers* reçoit les noms des fichiers de dictionnaires (sans extensions) installés sur le poste.
- *langNoms* reçoit les noms des langues exprimés dans la langue courante de l'application. Par exemple, pour un dictionnaire français, la valeur "français (France)" sera retournée sur une machine configurée en français et "French (France)" sur un système anglais. Le nom de la langue est suivi de "- Hunspell". Ce champ n'est valide que pour les fichiers "connus" de 4D. Pour les fichiers non connus (par exemple les fichiers personnalisés), le nom "N/A - Hunspell" est retourné. Ce principe n'empêche pas d'utiliser le dictionnaire (si le fichier concerné est valide), l'ID retourné pourra être passé à la commande **SPELL FIXER DICTIONNAIRE COURANT**.

Exemple

Vous avez placé "fr-classique+reforme1990.aff" et "fr-classique+reforme1990.dic" ainsi que "fr-dentiste.aff" et "fr-dentiste.dic" dans le répertoire Hunspell :

```
TABLEAU ENTIER LONG($langID;0)
TABLEAU TEXTE($dicName;0)
TABLEAU TEXTE($langDesc;0)
SPELL LIRE LISTE DICTIONNAIRES($langID;$dicName;$langDesc)
```

$langID	$dicName	$langDesc
65536	en_GB	anglais (Royaume-Uni)
65792	en_US	anglais (Etats-Unis)
131072	de_DE	allemand (Allemagne)
196608	es_ES	espagnol
262144	fr_FR	français (France)
589824	nb_NO	norvégien bokmal (Norvege)
1074036166	fr-classique+reforme1990	français (France) - Hunspell
1073901273	fr-dentiste	No description - Hunspell

SPELL VERIFIER TEXTE

SPELL VERIFIER TEXTE (leTexte ; posErr ; longErr ; posVérif ; tabSuggest)

Paramètre	Type		Description
leTexte	Texte	➡	Texte à vérifier
posErr	Entier long	⬅	Position du premier caractère du mot inconnu
longErr	Entier long	⬅	Longueur du mot inconnu
posVérif	Entier long	➡	Position de départ de la vérification
tabSuggest	Tableau texte	⬅	Liste des suggestions

Description

La commande **SPELL VERIFIER TEXTE** vérifie le contenu du paramètre *leTexte* à partir du caractère *posVérif* et retourne la position du premier mot inconnu rencontré (le cas échéant).

La commande retourne la position du premier caractère de ce mot dans *posErr* et sa longueur dans *longErr*. Le tableau *tabSuggest* reçoit la ou les suggestion(s) de correction proposée(s) par le correcteur orthographique.

Si la vérification démarre sans erreur et qu'un mot inconnu est rencontré, la variable système OK prend la valeur 0. Si une erreur d'initialisation se produit lors de la vérification ou si aucun mot n'est inconnu, OK prend la valeur 1.

Note OS X : Sous OS X lorsque le correcteur natif est activé, cette commande ne prend pas en charge la correction grammaticale.

Exemple

On souhaite compter le nombre de fautes potentielles dans un texte :

```
$pos:=1
$errCount:=0
TABLEAU TEXTE($tErrors;0)
TABLEAU TEXTE($tSuggestions;0)
Repeter
   SPELL VERIFIER TEXTE($myText;$errPos;$errLong;$pos;$tSuggestions)
   Si(OK=0)
      $errCount:=$errCount+1 // compteur de fautes
      $errorWord:=Sous chaine($myText;$errPos;$errLong)
      AJOUTER A TABLEAU($errors;$errorWord) // tableau des fautes
      $pos:=$errPos+$errLong  //poursuite de la vérification
   Fin de si
Jusque(OK=1)
 // Au final $errCount=Taille tableau($errorWord)
```

Dates et heures

- Ajouter a date
- Annee de
- Chaine heure
- Date
- Date du jour
- Heure
- Heure courante
- Jour de
- Mois de
- Nombre de millisecondes
- Nombre de ticks
- Numero du jour
- SIECLE PAR DEFAUT

Ajouter a date

Ajouter a date (date ; années ; mois ; jours) -> Résultat

Paramètre	Type		Description
date	Date	➡	Date à laquelle ajouter jours, mois et années
années	Entier long	➡	Nombre d'années à ajouter à la date
mois	Entier long	➡	Nombre de mois à ajouter à la date
jours	Entier long	➡	Nombre de jours à ajouter à la date
Résultat	Date	⮌	Date résultante

Description

Ajouter a date ajoute *années*, *mois* et *jours* à la date que vous avez passée dans *laDate*, et retourne la date résultante.

Alors que les **Opérateurs sur les dates** vous permettent d'ajouter des jours à une date, **Ajouter a date** vous permet d'ajouter rapidement des mois et des années sans vous soucier du nombre de jours par mois ou des années bissextiles (comme vous devriez le faire avec l'opérateur "+" sur les dates).

Exemple

```
 //Cette ligne calcule la date dans un an, le même jour
$vdDansUnAn:=Ajouter a date(Date du jour;1;0;0)

 //Cette ligne calcule la date le mois prochain, le même jour
$vdMoisProchain:=Ajouter a date(Date du jour;0;1;0)

 //Cette ligne fait la même chose que '$vdDemain:=Date du jour+1'
$vdDemain:=Ajouter a date(Date du jour;0;0;1)
```

Annee de

Annee de (date) -> Résultat

Paramètre	Type		Description
date	Date	⇒	Date dont vous voulez extraire l'année
Résultat	Entier long	⇐	Nombre indiquant l'année de date

Description

Annee de retourne un nombre indiquant l'année de *laDate*.

Exemple 1

L'exemple suivant illustre l'utilisation de **Annee de**. Les résultats sont assignés à la variable *Résultat* :

```
Résultat:=Annee de(!25/12/96!) ` Résultat prend la valeur 1996
Résultat:=Annee de(!25/12/1996!) ` Résultat prend la valeur 1996
Résultat:=Annee de(!25/12/1896!) ` Résultat prend la valeur 1896
Résultat:=Annee de(!25/12/2096!) ` Résultat prend la valeur 2096
Résultat:=Annee de(Date du jour) ` Résultat prend comme valeur l'année de la date d'aujourd'hui
```

Exemple 2

Reportez-vous à l'exemple de la fonction **Date du jour**.

Chaine heure

Chaine heure (secondes) -> Résultat

Paramètre	Type		Description
secondes	Entier long, Heure	➡	Secondes écoulées depuis minuit
Résultat	Chaîne	⮌	Heure sous forme de chaîne au format 24 heures

Description

La fonction **Chaine heure** retourne sous forme de chaîne alphanumérique sur 24 heures l'expression de type Heure passée dans *secondes*.

Le format appliqué à la chaîne est **HH:MM:SS**.

Si vous passez un nombre de secondes supérieur à celui qu'il y a dans un jour (86 400), **Chaine heure** continue d'ajouter les heures, les minutes et les secondes. Par exemple, **Chaine heure**(86401) retourne 24:00:01.

Note : Si vous voulez obtenir sous forme de chaîne une expression de type Heure dans des formats plus variés, utilisez la fonction **Chaine**.

Exemple

L'exemple suivant affiche une boîte de dialogue d'alerte avec le message “46800 secondes représentent 13:00:00” :

```
ALERTE("46800 secondes représentent "+Chaine heure(46800))
```

Date

Date (chaîneDate) -> Résultat

Paramètre	Type		Description
chaîneDate	Chaîne	⇒	Chaîne contenant la date à retourner
Résultat	Date	⇐	chaîneDate sous forme de Date

Description

La fonction **Date** extrait et retourne la date de la chaîne *chaîneDate*.

Le paramètre *chaîneDate* doit respecter soit le format date ISO, soit les paramètres régionaux du système.

Format Date ISO

La chaîne doit être formatée de la manière suivante : "AAAA-MM-JJ**T**HH:MM:SS", par exemple "2013-11-20T10:20:00". Dans ce cas, **Date** évaluera correctement *chaîneDate*, quels que soient les réglages de langue courants. Les décimales de secondes, précédées d'un point, sont prises en charge (ex : "2013-11-20T10:20:00.9854").

Si le format de *chaîneDate* ne respecte pas exactement ce schéma ISO, la date sera évaluée comme un format date court dépendant des paramètres régionaux du système.

Note : A compter de 4D v14, il est conseillé d'utiliser le format "AAAA-MM-JJTHH:MM:SS**Z**", conforme à la norme ISO et permettant d'exprimer le fuseau horaire.

Paramètres régionaux

Si *chaîneDate* ne correspond pas au format ISO, les paramètres régionaux définis dans le système d'exploitation pour une date courte sont utilisés pour l'évaluation. Par exemple, dans une version française de 4D, la date doit être par défaut de la forme JJ/MM/AA (jour, mois, année). Le jour et le mois peuvent être composés d'un ou deux chiffres. L'année peut être composée de deux ou quatre chiffres. Si l'année comporte deux chiffres, **Date** considère que la date appartient au XXe ou au XXIe siècle en fonction de la valeur saisie. Par défaut, la valeur pivot est 30 :

- si la valeur saisie est supérieure ou égale à 30, 4D considère que la date appartient au XXe siècle et ajoute 19 devant la valeur.
- si la valeur saisie est inférieure à 30, 4D considère que la date appartient au XXIe siècle et ajoute 20 devant la valeur.

Ce mécanisme peut être modifié à l'aide de la commande **SIECLE PAR DEFAUT**.

Les caractères de séparation de date autorisés sont les suivants : barre oblique (/), espace, point (.), virgule (,) et tiret (-).

Si une date invalide (telle que "13/35/94" ou "aa/12/94") est passée dans *chaîneDate*, **Date** retourne une date nulle (!00/00/00!). Il est de votre ressort de tester la validité de *chaîneDate*.

Exemple 1

L'exemple suivant demande à l'utilisateur de saisir une date. La chaîne saisie est convertie en date et stockée dans la variable DemDate :

```
DemDate:=Date(Demander("Saisissez une date :";Chaine(Date du jour)))
Si(OK=1)
  ` Faire quelque chose avec la date
Fin de si
```

Exemple 2

L'exemple suivant retourne la chaîne "25/12/97" sous forme de date :

```
vDate:=Date("25/12/97")
```

Exemple 3

Evaluation d'une date à partir d'une date au format ISO :

```
$vtDateISO:="2013-06-05T20:00:00"
$vDate:=Date($vtDateISO)
    //$vDate représente le 5 juin 2013 quelle que soit la langue du système
```

Date du jour

Date du jour {(*)} -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Retourne la date du jour du serveur
Résultat	Date	➲	Date du jour

Description

Date du jour retourne la date courante telle que définie dans l'horloge système de la machine.

4D Server : Si vous passez le paramètre astérisque (*) — lors d'une exécution sur un poste 4D Client —, la fonction retourne la date du jour telle que définie dans l'horloge du poste serveur.

Exemple 1

L'exemple suivant fait apparaître une boîte de dialogue d'alerte affichant la date du jour :

```
ALERTE("Nous sommes le "+Chaine(Date du jour)+".")
```

Exemple 2

Vous développez une application pour le marché international. Vous souhaitez savoir si la version de 4D avec laquelle votre application est exécutée fonctionne avec des dates formatées en MM/JJ/AAAA (version US) ou JJ/MM/AAAA (version française). Cette information est nécessaire pour vous permettre, par exemple, de personnaliser correctement les zones de saisie.

La méthode projet suivante vous permet de traiter cette question :

```
  ` Méthode projet (fonction) Format date système
  ` Format date système -> Chaine
  ` Format date système -> Format de données 4D par défaut

C_ALPHA(31;$0;$vsDate;$vsMJA;$vsMois;$vsJour;$vsAnnée)
C_ENTIER LONG($1;$vlPos)
C_DATE($vdDate)

  ` Récupérer une date dans laquelle les valeurs de mois, de jour et d'année sont toutes différentes
$vdDate:=Date du jour
Repeter
   $vsMois:=Chaine(Mois de($vdDate))
   $vsJour:=Chaine(Jour de($vdDate))
   $vsAnnée:=Chaine(Annee de($vdDate)%100)
   Si(($vsMois=$vsJour) | ($vsMois=$vsAnnée) | ($vsJour=$vsAnnée))
      OK:=0
      $vdDate:=$vdDate+1
   Sinon
      OK:=1
   Fin de si
Jusque(OK=1)
$0:=""   ` Initialisation du résultat de la fonction
$vsDate:=Chaine($vdDate)
$vlPos:=Position("/";$vsDate)   ` Trouver le premier séparateur / dans la chaîne ..
$vsMJA:=Sous chaine($vsDate;1;$vlPos-1)   ` Extraire les premiers chiffres de la date
$vsDate:=Sous chaine($vsDate;$vlPos+1)   ` Eliminer les premiers chiffres et le premier séparateur /
Au cas ou
   :($vsMJA=$vsMois)   ` Les chiffres expriment le mois
      $0:="MM"
   :($vsMJA=$vsJour)   ` Les chiffres expriment le jour
      $0:="JJ"
   :($vsMJA=$vsAnnée)   ` Les chiffres expriment l'année
      $0:="AAAA"
Fin de cas
$0:=$0+"/"   ` Commencer à construire le résultat de la fonction
$vlPos:=Position("/";$vsDate)   ` Trouver le deuxième séparateur dans la chaîne ../..
$vsMJA:=Sous chaine($vsDate;1;$vlPos-1)   ` Extraire les chiffres suivants de la date
$vsDate:=Sous chaine($vsDate;$vlPos+1)   ` Réduire la chaîne aux derniers chiffres de la date
Au cas ou
   :($vsMJA=$vsMois)   ` Les chiffres expriment le mois
      $0:=$0+"MM"
   :($vsMJA=$vsJour)   ` Les chiffres expriment le jour
      $0:=$0+"JJ"
   :($vsMJA=$vsAnnée)   ` Les chiffres expriment l'année
      $0:=$0+"AAAA"
Fin de cas
$0:=$0+"/"   ` Poursuivre la construction du résultat de la fonction
```

```
Au cas ou
   :($vsDate=$vsMois)    ` Les chiffres expriment le mois
      $0:=$0+"MM"
   :($vsDate=$vsJour)    ` Les chiffres expriment le jour
      $0:=$0+"DD"
   :($vsDate=$vsAnnée)    ` Les chiffres expriment l'année
      $0:=$0+"AAAA"
Fin de cas
  ` A ce moment, $0 vaut soit MM/JJ/AAAA soit JJ/MM/AAAA, ou encore...
```

Heure

Heure (valHeure) -> Résultat

Paramètre	Type		Description
valHeure	Chaîne, Entier long	➡	Valeur à retourner sous forme d'heure
Résultat	Heure	⮌	Heure définie par valHeure

Description

La fonction **Heure** retourne, sous la forme d'une expression de type Heure, l'heure définie dans le paramètre *valHeure*.

Le paramètre *valHeure* peut contenir soit :

- une chaîne contenant une heure exprimée dans l'un des formats d'heure standard de 4D correspondant à la langue de votre système (pour plus d'informations, reportez-vous à la description de la commande **Chaine**).
- un entier long représentant un nombre de secondes écoulées depuis 00:00:00.

Exemple 1

L'exemple suivant affiche une boîte de dialogue d'alerte avec le message “1:00 P.M. = 13 heures 0 minute.” :

```
ALERTE("1:00 P.M. = "+Chaine(Heure("13:00:00");Heures minutes))
```

Exemple 2

Vous pouvez exprimer toute valeur numérique sous forme d’heure :

```
vHeure:=Heure(10000)
    //vHeure vaut 02:46:40
vHeure2:=Heure((60*60)+(20*60)+5200)
    //vHeure2 vaut 02:46:40
```

Heure courante

Heure courante {(*)} -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Retourne l'heure courante sur le poste serveur
Résultat	Heure	⇐	Heure courante

Description

La fonction **Heure courante** retourne l'heure courante définie dans l'horloge de votre système.

L'heure courante est toujours comprise entre *00:00:00* et *23:59:59*. Vous pouvez utiliser les fonctions **Chaine** ou **Chaine heure** pour convertir en chaîne alphanumérique l'expression de type heure retournée par **Heure courante**.

4D Server : Si vous passez le paramètre astérisque (*) — lors d'une exécution sur un poste 4D Client —, la fonction retourne l'heure courante telle que définie dans l'horloge du poste serveur.

Exemple 1

L'exemple suivant vous permet de mesurer la durée d'une opération. Dans cet exemple, vous voulez chronométrer la méthode ***longueOpération*** :

```
CelaPrend:=Heure courante ` Stocker l'heure de départ
longueOpération ` Effectuer l'opération
ALERTE("L'opération a pris "+Chaine(Heure courante-CelaPrend)) ` Afficher la durée
```

Exemple 2

L'exemple suivant extrait les heures, minutes et secondes de l'heure courante :

```
$vhMaintenant:=Heure courante
ALERTE("L'heure courante est : "+Chaine($vhMaintenant\3600))
ALERTE("La minute courante est : "+Chaine(($vhMaintenant\60)%60))
ALERTE("La seconde courante est : "+Chaine($vhMaintenant%60))
```

Jour de

Jour de (date) -> Résultat

Paramètre	Type		Description
date	Date	⇒	Date dont vous voulez extraire le jour
Résultat	Entier long	⇐	Jour du mois de date

Description

Jour de retourne le jour du mois de *laDate*.

Note : Jour de retourne une valeur entre 1 et 31. Pour obtenir le numéro du jour de la semaine pour une date, vous devez utiliser la commande **Numero du jour**.

Exemple 1

L'exemple suivant illustre l'utilisation de **Jour de**. Les valeurs retournées sont stockées dans la variable *Résultat*. Les commentaires décrivent la valeur de *Résultat* :

```
Résultat:=Jour de(!25/12/96!) ` Résultat vaut 25
Résultat:=Jour de(Date du jour) ` Résultat prend la valeur du jour d'aujourd'hui
```

Exemple 2

Reportez-vous à l'exemple de la fonction **Date du jour**.

Mois de

Mois de (laDate) -> Résultat

Paramètre	Type		Description
laDate	Date	➡	Date dont vous voulez extraire le mois
Résultat	Entier long	➲	Nombre indiquant le mois de date

Description

Mois de retourne un nombre représentant le numéro du mois de *laDate*.

Note : C'est le numéro du mois est retourné, et non son nom (reportez-vous à l'exemple ci-dessous).

Pour comparer la valeur retournée par cette fonction, 4D fournit les constantes prédéfinies suivantes, placées dans le thème "**Jours et mois**" :

Constante	Type	Valeur
Janvier	Entier long	1
Février	Entier long	2
Mars	Entier long	3
Avril	Entier long	4
Mai	Entier long	5
Juin	Entier long	6
Juillet	Entier long	7
Août	Entier long	8
Septembre	Entier long	9
Octobre	Entier long	10
Novembre	Entier long	11
Décembre	Entier long	12

Exemple 1

L'exemple suivant illustre l'utilisation de **Mois de**. Les valeurs retournées sont assignées à la variable Résultat. Les commentaires fournissent les valeurs de Résultat :

```
Résultat:=Mois de(!25/12/96!) ` Résultat vaut 12
Résultat:=Mois de(Date du jour) ` Résultat prend la valeur du mois d'aujourd'hui
```

Exemple 2

Reportez-vous à l'exemple de la fonction **Date du jour**.

Nombre de millisecondes

Nombre de millisecondes -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Nombre de millisecondes (1000ème de seconde) écoulées depuis le démarrage de la machine

Description

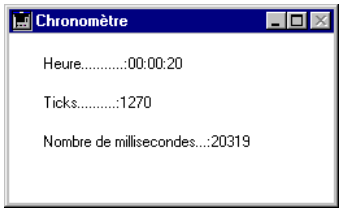
Nombre de millisecondes retourne le nombre de millisecondes (1 milliseconde = 1/1000ème de seconde) écoulées depuis le démarrage de la machine.

Exemple

Le code suivant :

```
Creer fenetre(100;100;350;230;0;"Chronomètre")
$vhDébutHeure:=Heure courante
$vlDébutTicks:=Nombre de ticks
$vrDébutMillisecondes:=Nombre de millisecondes
Repeter
   POSITION MESSAGE(2;1)
   MESSAGE("Heure...........:"+Chaine(Heure courante-$vhDébutHeure))
   POSITION MESSAGE(2;3)
   MESSAGE("Ticks..........:"+Chaine(Nombre de ticks-$vlDébutTicks))
   POSITION MESSAGE(2;5)
   MESSAGE("Nombre de millisecondes...:"+Chaine(Nombre de millisecondes-$vrDébutMillisecondes))
Jusque((Heure courante-$vhDébutHeure)>=?00:01:00?)
FERMER FENETRE
```

... affiche la fenêtre suivante pendant une minute :



Nombre de ticks

Nombre de ticks -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Nombre de ticks (60ème de seconde) écoulés depuis le démarrage de la machine

Description

Nombre de ticks retourne le nombre de ticks (1 tick = 1/60ème de seconde) écoulés depuis le démarrage de la machine.
Note : Nombre de ticks retourne une valeur de type Entier long.

Exemple

Référez-vous à l'exemple de la fonction **Nombre de millisecondes**.

Numero du jour

Numero du jour (laDate) -> Résultat

Paramètre	Type		Description
laDate	Date	⇒	Date dont vous souhaitez connaître le numéro du jour
Résultat	Entier long	⇐	Numéro représentant le jour de la semaine auquel date correspond

Description

La fonction **Numero du jour** retourne un numéro représentant le jour de la semaine auquel *laDate* correspond.

Note : Si une date nulle est passée à **Numero du jour**, la fonction retourne 2.

4D fournit les constantes prédéfines suivantes, placées dans le thème "**Jours et mois**" :

Constante	Type	Valeur
Dimanche	Entier long	1
Lundi	Entier long	2
Mardi	Entier long	3
Mercredi	Entier long	4
Jeudi	Entier long	5
Vendredi	Entier long	6
Samedi	Entier long	7

Note : Numero du jour retourne une valeur comprise entre 1 et 7. Pour obtenir le numéro du jour dans le sens "date du mois", utilisez la fonction **Jour de**.

Exemple

L'exemple suivant est une fonction qui retourne le jour d'aujourd'hui sous forme de chaîne :

```
$Jour :=Numero du jour(Date du jour) ` $Jour prend comme valeur le numéro du jour courant
Au cas ou
   :($Jour =1)
      $0:="Dimanche"
   :($Jour =2)
      $0:="Lundi"
   :($Jour =3)
      $0:="Mardi"
   :($Jour =4)
      $0:="Mercredi"
   :($Jour =5)
      $0:="Jeudi"
   :($Jour =6)
      $0:="Vendredi"
   :($Jour =7)
      $0:="Samedi"
Fin de cas
```

SIECLE PAR DEFAUT

SIECLE PAR DEFAUT (siècle {; anPivot})

Paramètre	Type		Description
siècle	Entier long	➡	Siècle par défaut (moins un) lors de la saisie d'années sur 2 chiffres
anPivot	Entier long	➡	Année pivot lors de la saisie d'années sur 2 chiffres

Description

La commande **SIECLE PAR DEFAUT** vous permet de définir le siècle courant par défaut et l'année pivot adoptés par 4D lorsque des dates sont saisies avec seulement deux chiffres pour l'année.

L'année pivot indique la valeur au-dessous de laquelle une année saisie sur deux chiffres sera interprétée comme appartenant au siècle suivant :

- si l'année saisie est supérieure ou égale à l'année pivot, elle appartient au siècle courant,
- si l'année saisie est inférieure à l'année pivot, elle appartient au siècle suivant (relativement au siècle courant par défaut).

Par défaut, 4D présume que les dates appartiennent au 20e siècle et utilise la valeur 30 comme année pivot :

- Si vous saisissez la date 25/01/97, 4D considère que vous voulez indiquer le 25 janvier 1997
- Si vous saisissez la date 25/01/30, 4D considère que vous voulez indiquer le 25 janvier 1930
- Si vous saisissez la date 25/01/29, 4D considère que vous voulez indiquer le 25 janvier 2029
- Si vous saisissez la date 25/01/07, 4D considère que vous voulez indiquer le 25 janvier 2007

La commande **SIECLE PAR DEFAUT** permet de modifier ce comportement par défaut. Une fois exécutée, elle prend effet immédiatement.
Vous pouvez passer uniquement un siècle par défaut, ou un siècle par défaut et une année pivot.

Si vous passez uniquement un nouveau siècle par défaut moins un dans *siècle*, 4D interprétera toutes les années saisies sur deux chiffres comme appartenant à ce siècle.
Par exemple, après l'appel :

```
SIECLE PAR DEFAUT(20) ` Fixer le 21e siècle comme siècle par défaut
```

- Si vous saisissez la date 25/01/97, 4D considère que vous voulez indiquer le 25 janvier 2097
- Si vous saisissez la date 25/01/07, 4D considère que vous voulez indiquer le 25 janvier 2007

En outre, vous pouvez spécifier le paramètre *anPivot*.
Par exemple, après l'appel :

```
SIECLE PAR DEFAUT(19;95) ` Fixer le 21e siècle comme siècle par défaut si l'année est inférieure à 95
```

- Si vous saisissez la date 25/01/97, 4D considère que vous voulez indiquer le 25 janvier 1997
- Si vous saisissez la date 25/01/07, 4D considère que vous voulez indiquer le 25 janvier 2007

Notez que cette commande n'affecte que l'interprétation des dates lorsque les années sont saisies sur 2 chiffres. Quels que soient les paramètres passés à **SIECLE PAR DEFAUT** :

- Si vous saisissez la date 25/01/1997, 4D considère que vous voulez indiquer le 25 janvier 1997
- Si vous saisissez la date 25/01/2097, 4D considère que vous voulez indiquer le 25 janvier 2097
- Si vous saisissez la date 25/01/1907, 4D considère que vous voulez indiquer le 25 janvier 1907
- Si vous saisissez la date 25/01/2007, 4D considère que vous voulez indiquer le 25 janvier 2007

Cette commande affecte uniquement la saisie des données. Elle n'influe pas sur le stockage des données, les calculs, etc.

Définition structure

- Commandes du thème Définition structure
- Champ
- CREER INDEX
- Est un numero de champ valide
- Est un numero de table valide
- EXPORTER STRUCTURE
- FIXER CHEMIN DONNEES EXTERNES
- FIXER INDEX
- IMPORTER STRUCTURE
- Lire chemin donnees externes
- LIRE NOMS TABLES MANQUANTES
- Lire numero dernier champ
- Lire numero derniere table
- LIRE PROPRIETES CHAMP
- LIRE PROPRIETES LIEN
- LIRE PROPRIETES SAISIE CHAMP
- LIRE PROPRIETES TABLE
- Nom de la table
- Nom du champ
- REACTIVER INDEX
- RECHARGER DONNEES EXTERNES
- REGENERER TABLE MANQUANTE
- SUPPRIMER INDEX
- SUSPENDRE INDEX
- Table

Commandes du thème Définition structure

Les commandes de ce thème retournent la description de la structure de la base. Elles permettent de connaître le nombre de tables, le nombre de champs dans chaque table, les noms des tables et des champs, ainsi que le type et les propriétés de chaque champ.

L'identification précise de la structure de la base est très utile quand vous développez et utilisez des groupes de méthodes projets et formulaires qui peuvent être copiées dans différentes bases.

La possibilité de lire la structure de la base vous permet de développer et d'utiliser du code portable.

Note : Il est possible de créer et de modifier des tables et des champs 4D par programmation, à l'aide des commandes du moteur SQL intégré de 4D, comme **CREATE TABLE** ou **ALTER TABLE**. Pour plus d'informations, reportez-vous au manuel "**4D - Référence SQL**".

Compter les tables et les champs

Il est possible de supprimer des tables et des champs 4D. Cette possibilité doit être prise en compte dans les algorithmes utilisés pour dénombrer les tables et les champs. Il est nécessaire d'utiliser des algorithmes combinant les commandes **Lire numero derniere table** et **Lire numero dernier champ**, **Est un numero de table valide** et **Est un numero de champ valide**. Voici un exemple de ce type d'algorithme :

```
Boucle($latable;1;Lire numero derniere table)
   Si(Est un numero de table valide($latable))
      Boucle($lechamp;1;Lire numero dernier champ($latable))
         Si(Est un numero de champ valide($latable;$lechamp))
            ... `Le champ existe et est valide
         Fin de si
      Fin de boucle
   Fin de si
Fin de boucle
```

Champ

Champ (numTable ; numChamp) -> ptrChamp

Paramètre	Type		Description
numTable	Entier long	→	Numéro de table
numChamp	Entier long	→	Numéro de champ
ptrChamp	Pointeur	↩	Pointeur de champ

Champ (ptrChamp) -> numChamp

Paramètre	Type		Description
ptrChamp	Pointeur	→	Pointeur de champ
numChamp	Entier long	↩	Numéro de champ

Description

La commande **Champ** a deux syntaxes :

- Si vous passez un numéro de table dans *numTable* et un numéro de champ dans *numChamp*, **Champ** retourne un pointeur vers le champ.
- Si vous passez un pointeur vers un champ dans *ptrChamp*, **Champ** retourne le numéro du champ.

Exemple 1

L'exemple suivant assigne la variable *ChampPtr* à un pointeur vers le deuxième champ de la troisième table :

```
ChampPtr:=Champ(3;2)
```

Exemple 2

Si vous passez *champPtr* (un pointeur vers le 2e champ de la table) à **Champ**, la valeur 2 est retournée. La ligne suivante assigne la valeur 2 à *champNum* :

```
champNum:=Champ(champPtr)
```

Exemple 3

Dans l'exemple, la variable *champNum* est égale au numéro de champ de [Table3]Champ2 :

```
champNum:=Champ(->[Table3]Champ2)
```

CREER INDEX

CREER INDEX (laTable ; tabChamps ; typeIndex ; nomIndex {; *})

Paramètre	Type		Description
laTable	Table	⇒	Table pour laquelle créer un index
tabChamps	Tableau pointeur	⇒	Pointeur(s) vers le(s) champ(s) à indexer
typeIndex	Entier long	⇒	Type d'index à créer : -1 = Mots-clés, 0 = par défaut, 1 = B-Tree standard, 3 = BTree cluster
nomIndex	Texte	⇒	Nom de l'index à créer
*	Opérateur	⇒	Si passé = indexation asynchrone

Description

La commande **CREER INDEX** permet de créer :

- un index standard sur un ou plusieurs champs (index composite) ou
- un index de mots-clés sur un champ.

L'index est créé pour la table *laTable* en utilisant le ou les champ(s) désigné(s) par le tableau de pointeurs *tabChamps*. Ce tableau contient une seule ligne si vous souhaitez créer un index simple et deux ou plusieurs lignes si vous souhaitez créer un index composite (sauf index de mots-clés). Dans le cas d'index composites, l'ordre des champs dans le tableau est important lors de la construction de l'index.

Le paramètre *typeIndex* vous permet de définir le type d'index à créer. Vous pouvez passer une des constantes suivantes, placées dans le thème **Type index** :

Constante	Type	Valeur	Comment
Index BTree cluster	Entier long	3	Index de type B-Tree utilisant des clusters. Ce type d'index est optimisé lorsque l'index contient peu de clés, c'est-à-dire lorsque les mêmes valeurs reviennent souvent dans les données.
Index BTree standard	Entier long	1	Index de type B-Tree classique. Ce type d'index polyvalent est utilisé dans les versions précédentes de 4D
Index de mots clés	Entier long	-1	Permet l'indexation mot à mot du contenu du champ. Ce type d'index est utilisable avec les champs de type Texte, Alpha et Image. Attention, les index de mots-clés ne peuvent pas être composites.
Type index par défaut	Entier long	0	4D définit le type d'index (hors index de mots-clés) le plus optimisé en fonction du contenu du champ.

Note : Un index B-Tree associé à un champ de type texte stocke au maximum les 1024 premiers caractères du champ. Par conséquent dans ce contexte, les recherches sur des chaînes contenant plus de 1024 caractères ne pourront aboutir.

Passez dans *nomIndex* le nom de l'index à créer. Nommer les index est nécessaire si plusieurs index de types différents peuvent être associés à un même champ et si vous souhaitez pouvoir les supprimer individuellement à l'aide de la commande **SUPPRIMER INDEX**. Si l'index *nomIndex* existe déjà, la commande ne fait rien.

Le paramètre facultatif *, lorsqu'il est passé, permet d'effectuer l'indexation en mode asynchrone. Dans ce mode, la méthode d'origine poursuit son exécution après l'appel de la commande, que l'indexation soit terminée ou non.

Si la commande **CREER INDEX** rencontre des enregistrements verrouillés, elle ne les indexe pas et attend qu'ils soient libérés.

Si une erreur se produit durant l'exécution de la commande (champ non indexable, tentative de création d'index de mots-clés sur plusieurs champs, etc.), une erreur est générée. Cette erreur peut être interceptée à l'aide d'une méthode d'appel sur erreur.

Exemple 1

Création de deux index standard sur les champs "Nom" et "Téléphone" de la table [Clients] :

```
TABLEAU POINTEUR(tabPtrChp;1)
tabPtrChp{1}:=->[Clients]Nom
CREER INDEX([Clients];tabPtrChp;Index BTree standard;"IdxCltNom")
tabPtrChp{1}:=->[Clients]Téléphone
CREER INDEX([Clients];tabPtrChp;Index BTree standard;"IdxCltTel")
```

Exemple 2

Création d'un index de mots-clés sur le champ "Observations" de la table [Clients] :

```
TABLEAU POINTEUR(tabPtrChp;1)
tabPtrChp{1}:=->[Clients]Observations
CREER INDEX([Clients];tabPtrChp;Index de mots clés;"IdxCltObs")
```

Exemple 3

Création d'un index composite sur les champs "CodePostal" et "Ville" de la table [Clients] :

```
TABLEAU POINTEUR(tabPtrChp;2)
tabPtrChp{1}:=->[Clients]CodePostal
tabPtrChp{2}:=->[Clients]Ville
CREER INDEX([Clients];tabPtrChp;Index BTree standard;"CPVille")
```

Est un numero de champ valide

Est un numero de champ valide (numTable | ptrTable ; numChamp) -> Résultat

Paramètre	Type		Description
numTable | ptrTable	Entier long, Pointeur	⇒	Numéro de table ou Pointeur vers une table
numChamp	Entier long	⇒	Numéro de champ
Résultat	Booléen	⇐	Vrai = le champ existe dans la table, Faux = le champ n'existe pas dans la table

Description

La commande **Est un numero de champ valide** retourne Vrai si le champ dont le numéro est passé dans *numChamp* existe dans la table dont le numéro ou le pointeur est passé dans le paramètre *numTable* ou *ptrTable*. Si le champ n'existe pas, la commande retourne Faux. A noter que la commande retourne Faux si la table du champ se trouve dans la Corbeille de l'Explorateur.

Cette commande permet de détecter d'éventuelles suppressions de champs, ce qui crée des ruptures dans la séquence des numéros de champs.

Est un numero de table valide

Est un numero de table valide (numTable) -> Résultat

Paramètre	Type		Description
numTable	Entier long	⇒	Numéro de table
Résultat	Booléen	⇐	Vrai = la table existe dans la base, Faux = la table n'existe pas dans la base

Description

La commande **Est un numero de table valide** retourne Vrai si la table dont le numéro est passé dans *numTable* existe dans la base et Faux sinon. A noter que la commande retourne Faux si la table se trouve dans la Corbeille de l'Explorateur.

Cette commande permet de détecter d'éventuelles suppressions de tables, ce qui crée des ruptures dans la séquence des numéros de tables.

EXPORTER STRUCTURE

EXPORTER STRUCTURE (structureXML)

Paramètre	Type		Description
structureXML	Variable texte	⇦	Export de la définition XML de la structure de la base 4D

Description

La commande **EXPORTER STRUCTURE** exporte dans *structureXML* la définition de la structure de la base 4D courante au format XML. Cette commande utilise les mêmes mécanismes que la commande de menu **Exporter > Définition de structure vers le fichier XML...** disponible dans l'interface du mode Développement de 4D (voir **Exporter et importer des définitions de structure**).

Passez dans *xmlStructure* la variable texte destinée à stocker la définition de la structure. La définition exportée inclut les tables, les champs, les index et les liens, ainsi que leurs attributs et toutes les caractéristiques nécessaires à la description complète de la structure. Les éléments invisibles sont exportés avec l'attribut correspondant. Les éléments supprimés, cependant, ne sont pas exportés.

La "grammaire" interne des définitions de structure 4D est documentée via des fichiers DTD — également utilisés pour la validation des fichiers XML. Les fichiers DTD utilisés par 4D sont regroupés dans le dossier **DTD**, situé à côté de l'application 4D. Les fichiers **base_core.dtd** et **common.dtd** sont utilisés pour les définitions de structure. Pour plus d'informations, n'hésitez pas à les consulter ainsi que les commentaires qu'ils contiennent.

Exemple

Vous voulez exporter la structure de la base courante dans un fichier texte :

```
C_TEXTE($vTStruc)
EXPORTER STRUCTURE($vTStruc)
TEXTE VERS DOCUMENT("myStructure.xml";$vTStruc)
```

FIXER CHEMIN DONNEES EXTERNES

FIXER CHEMIN DONNEES EXTERNES (leChamp ; chemin)

Paramètre	Type		Description
leChamp	Texte, BLOB, Image, Objet	⇒	Champ pour lequel définir le lieu de stockage
chemin	Texte, Entier long	⇒	Chemin d'accès et nom du fichier de stockage externe ou 0 = utiliser la définition en structure 1 = utiliser le dossier par défaut

Description

La commande **FIXER CHEMIN DONNEES EXTERNES** permet de définir ou de modifier, pour l'enregistrement courant, l'emplacement de stockage externe du champ *leChamp* passé en paramètre.

4D autorise le stockage des données des champs de type Texte, Blob et Image et Objet *à l'extérieur* du fichier de données. Pour une description complète de cette fonctionnalité, reportez-vous au manuel *Mode Développement*.

Le paramétrage défini par cette commande sera appliqué uniquement lors du stockage sur disque de l'enregistrement courant. Si l'enregistrement courant est annulé, la commande ne fait rien. Les paramètres de stockage définis dans la structure de l'application ne sont pas modifiés.

Vous pouvez passer dans *chemin* soit un chemin d'accès personnalisé, soit une constante désignant un emplacement automatique :

- **chemin d'accès personnalisé au fichier**
  Dans ce cas, vous utilisez le stockage externe en "mode personnalisé". Dans ce mode, certaines fonctions de base de données de 4D ne sont pas disponibles automatiquement (cf. manuel *Mode Développement*). En particulier, vous devez gérer vous-même la création ou la modification des fichiers.
  Vous pouvez passer un chemin relatif au fichier de données ou un chemin absolu, incluant obligatoirement le nom du fichier de stockage et son extension. Vous devez utiliser la syntaxe système. Pour définir un chemin relatif au fichier de données, passez "../" (Windows) ou "..:" (OS X) au début de la chaîne. Vous pouvez désigner tout dossier, y compris le dossier par défaut des fichiers externes de la base (*nomBase*.ExternalData) - dans ce cas, les fichiers seront inclus lors de la sauvegarde de la base.
  Le fichier désigné par le paramètre *chemin* doit exister et être accessible au moment de l'exécution de la commande. A noter que si le chemin est invalide (fichier ou dossier manquant), une erreur est retournée uniquement dans le cas où *chemin* a été défini en absolu. Dans le cas où *chemin* a été défini en relatif, vous devez vous assurer de sa validité, aucune erreur n'est générée si le fichier n'est pas trouvé.
  Si vous stockez le fichier externe dans le même dossier que le fichier de données ou un de ses sous-dossiers, 4D considérera que le chemin défini est relatif au fichier de données et maintiendra le lien même si le dossier du fichier de données est déplacé ou renommé.
  A noter que ce principe permet de "partager" un même fichier externe entre plusieurs enregistrements. Toute modification effectuée sur ce fichier externe est disponible dans tous les enregistrements. Attention dans ce cas, si plusieurs process peuvent écrire simultanément les mêmes champs, vous devez empêcher les accès concurrents via des sémaphores afin de ne pas risque d'endommager les fichiers externes.

- **emplacement automatique**
  Vous pouvez désigner deux emplacements automatiques, à l'aide des constantes suivantes, placées dans le thème **Maintenance fichier de données** :

Constante	Type	Valeur	Comment
Utiliser définition structure	Entier long	0	4D utilisera les paramètres définis dans la structure pour le stockage du champ (cf. manuel *Mode Développement*). Si vous passez d'un stockage externe à un stockage interne, le fichier externe n'est pas supprimé.
Utiliser dossier par défaut	Entier long	1	Les données du champ passé en paramètre seront stockées dans le dossier par défaut, nommé *nomBase*.ExternalData et placé à côté du fichier de données. Dans ce mode, les données externes sont gérées par 4D comme si elles étaient à l'intérieur du fichier de données.

Une fois la commande exécutée, 4D maintient automatiquement le lien entre le champ de l'enregistrement et le fichier sur disque, il n'est pas nécessaire de réexécuter la commande (hormis si le *chemin* doit être modifié). Si 4D ne peut plus accéder aux données du champ (fichier de stockage renommé ou supprimé, chemin modifié...), le champ est vide mais aucune erreur n'est générée.

Note : La commande **FIXER CHEMIN DONNEES EXTERNES** peut uniquement être exécutée sur 4D local ou 4D Server. Si elle est exécutée sur un 4D distant, elle ne fait rien.

Exemple

Vous souhaitez enregistrer dans le champ image un fichier existant, stocké à l'extérieur des données, dans le dossier de la base :

```
CREER ENREGISTREMENT([Photos])
[Photos]Nom:="Paris.png"
FIXER CHEMIN DONNEES EXTERNES([Photos]Vignette;Dossier 4D(Dossier base)+"custom"+Séparateur dossier+[Photos]Nom)
 //"/custom/Paris.png" doit exister à côté du fichier de structure
STOCKER ENREGISTREMENT([Photos])
```

FIXER INDEX

FIXER INDEX (leChamp ; index {; mode} {; *})

Paramètre	Type		Description
leChamp	Champ	⇒	Champ duquel créer ou supprimer l'index
index	Booléen, Entier	⇒	• Vrai=Créer l'index, Faux=Supprimer l'index, ou • Créer un index de type : -1=mots-clés, 0=par défaut, 1=B-Tree standard, 3=B-Tree cluster
mode	Entier long	⇒	Obsolète (paramètre ignoré)
*		⇒	Indexation asynchrone si * est passé

Description

Note de compatibilité : Cette commande est conservée pour des raisons de compatibilité uniquement. Il est désormais recommandé d'utiliser les commandes **CREER INDEX** et **SUPPRIMER INDEX** pour gérer les index par programmation.

La commande **FIXER INDEX** admet deux syntaxes :

- Si vous passez un booléen dans le paramètre *index*, la commande crée ou supprime l'index du champ que vous avez passé dans *leChamp*.
- Si vous passez un entier dans le paramètre *index*, la commande crée un index du type spécifié.

index = booléen
Pour indexer le champ, passez Vrai dans *index*. La commande crée un index du type par défaut. Si l'index existe déjà, la commande ne fait rien.
Si vous passez Faux dans *index*, la commande supprimera tous les index standard (c'est-à-dire, non composites et non mots-clés) associés au champ. S'il n'existe pas d'index, la commande ne fait rien.

index = entier
Dans ce cas, la commande crée un index du type spécifié pour *leChamp*. Vous pouvez passer une des constantes suivantes, placées dans le thème "**Type index**" :

Constante	Type	Valeur	Comment
Index BTree cluster	Entier long	3	Index de type B-Tree utilisant des clusters. Ce type d'index est optimisé lorsque l'index contient peu de clés, c'est-à-dire lorsque les mêmes valeurs reviennent souvent dans les données.
Index BTree standard	Entier long	1	Index de type B-Tree classique. Ce type d'index polyvalent est utilisé dans les versions précédentes de 4D
Index de mots clés	Entier long	-1	Permet l'indexation mot à mot du contenu du champ. Ce type d'index n'est utilisable qu'avec les champs de type Texte ou Alpha. Attention, les index de mots-clés ne peuvent pas être composites.
Type index par défaut	Entier long	0	4D définit le type d'index (hors index de mots-clés) le plus optimisé en fonction du contenu du champ.

Note : Un index B-Tree associé à un champ de type texte stocke au maximum les 1024 premiers caractères du champ. Par conséquent dans ce contexte, les recherches sur des chaînes contenant plus de 1024 caractères ne pourront aboutir.

FIXER INDEX n'indexera pas les enregistrements verrouillés ; la commande attendra que les enregistrements soient libérés.

Depuis la version 11, le paramètre *mode* est inutile et est ignoré s'il est passé.

Le paramètre optionnel * indique une indexation asynchrone (simultanée). Une indexation asynchrone permet à la méthode appelante de poursuivre son exécution immédiatement après l'appel, que l'indexation soit terminée ou non. Cependant, l'exécution sera stoppée si une commande requiert l'index.

Notes :

- Les index créés par cette commande ne portent pas de nom. Ils ne pourront pas être supprimés par la commande **SUPPRIMER INDEX** via la syntaxe basée sur le nom.
- Cette commande ne permet pas de créer ou de supprimer des index composites.
- Cette commande ne permet pas de supprimer un index de mots-clés créé par la commande **CREER INDEX**.

Exemple 1

L'exemple suivant indexe le champ *[Clients]Num* :

```
LIBERER ENREGISTREMENT([Clients])
FIXER INDEX([Clients]Num;Vrai)
```

Exemple 2

Vous souhaitez indexer le champ [Clients]Nom, en mode asynchrone :

```
FIXER INDEX([Clients]Nom;Vrai;*)
```

Exemple 3

Création d'un index de mots-clés :

```
FIXER INDEX([Livres]Résumé;Index de mots clés)
```

IMPORTER STRUCTURE

IMPORTER STRUCTURE (structureXML)

Paramètre	Type		Description
structureXML	Texte	⇨	Définition XML de la structure de la base 4D

Description

La commande **IMPORTER STRUCTURE** vous permet d'importer, dans la base courante, la définition XML de la structure de la base 4D passée dans le paramètre *structureXML*.

Le paramètre *structureXML* doit contenir une définition valide de structure 4D au format XML. Pour obtenir ce type de définition, vous pouvez utiliser l'un des moyens suivants :

- exécuter la nouvelle commande ,
- sélectionner la commande de menu **Exporter > Définition de structure vers le fichier XML...** disponible dans l'interface du mode Développement de 4D (voir **Exporter et importer des définitions de structure**),
- créer ou modifier un fichier XML personnalisé basé sur les DTD publiques présentes dans le dossier "DTD" de l'application 4D.

La définition de structure importée est ajoutée à la structure déjà ouverte et est affichée dans l'éditeur de Structure standard de 4D parmi les tables existantes (s'il y en a). Si une table importée a le même nom qu'une table locale, une erreur est générée et l'opération d'import est annulée.

Vous pouvez également importer la structure dans une base vide, et ainsi créer une nouvelle base.

Une erreur est générée si la structure est en mode compilé ou en lecture seulement.

Cette commande ne peut pas être appelée depuis une application 4D en mode distant.

Exemple

Vous souhaitez importer une définiton de structure stockée sur disque dans la base courante :

```
$struc:=Document vers texte("c:\\4DStructures\\Employee.xml")
IMPORTER STRUCTURE($struc)
```

Lire chemin donnees externes

Lire chemin donnees externes (leChamp) -> Résultat

Paramètre	Type		Description
leChamp	Texte, BLOB, Image	⇒	Champ dont vous souhaitez obtenir le lieu de stockage
Résultat	Texte	⇐	Chemin d'accès complet du fichier de stockage externe

Description

La commande **Lire chemin donnees externes** retourne le chemin d'accès complet du fichier de stockage externe des données du champ passé dans le paramètre *leChamp*, pour l'enregistrement courant. Le champ passé en paramètre doit être de type Texte, Blob ou Image. La commande retourne le chemin d'accès du fichier de stockage même si le fichier n'existe pas ou n'est pas accessible.

Cette commande vous permet notamment de recopier le fichier externe.

Note : Pour plus d'informations sur le stockage externe de données, reportez-vous au manuel *Mode Développement*.

Cette commande retourne une chaîne vide dans les cas suivants :

- le champ n'est pas stocké en-dehors du fichier de données,
- le champ a une valeur Null (et ne contient pas de chemin d'accès),
- la commande est exécutée depuis un 4D distant.

LIRE NOMS TABLES MANQUANTES

LIRE NOMS TABLES MANQUANTES (tabManquantes)

Paramètre	Type		Description
tabManquantes	Tableau texte	⇦	Noms des tables manquantes dans la base

Description

La commande **LIRE NOMS TABLES MANQUANTES** retourne dans le tableau *tabManquantes* les noms de toutes les tables manquantes de la base courante.

Les tables manquantes sont des tables dont les données sont présentes dans le fichier de données mais qui n'existent pas au niveau de la structure courante. Ce cas se produit lorsqu'un fichier de données est ouvert avec des versions différentes de la structure.

Typiquement, le scénario est le suivant :

- le développeur fournit une structure contenant les tables A, B et C,
- l'utilisateur ajoute des tables personnalisées D et E à l'aide, par exemple, des commandes **SQL** intégrées de 4D, et stocke des données dans ces tables,
- le développeur fournit une nouvelle version de la structure. Elle ne contient pas les tables D et E.
  Dans ce cas, la version utilisateur de la base contient toujours les données des tables D et E, mais elles ne sont plus accessibles. La commande **LIRE NOMS TABLES MANQUANTES** retournera les noms "D" et "E".

Une fois que vous avez identifié les tables manquantes de la base, vous pouvez les réactiver via la commande **REGENERER TABLE MANQUANTE**.

Note : Les données des tables manquantes sont effacées en cas de compactage du fichier de données (si les tables n'ont pas été regénérées entre-temps).

Lire numero dernier champ

Lire numero dernier champ (numTable | ptrTable) -> Résultat

Paramètre	Type		Description
numTable | ptrTable	Entier long, Pointeur	➡	Numéro de table ou Pointeur vers une table
Résultat	Entier long	⮌	Numéro de champ le plus élevé dans la table

Description

La commande **Lire numero dernier champ** retourne le numéro de champ le plus élevé parmi les champs de la table dont le numéro ou le pointeur est passé dans le paramètre *numTable* ou *ptrTable*.

Les champs sont numérotés dans l'ordre où ils ont été créés. Si aucun champ n'a été supprimé dans la table, cette commande retourne donc le nombre de champs que contient la table. Dans le cadre de boucles itératives sur les numéros de champs de la table, vous devez utiliser la commande **Est un numero de champ valide** afin de vérifier que le champ n'a pas été supprimé.

Exemple

La méthode projet suivante crée le tableau *taChamps* avec les noms des champs de la table dont le pointeur est reçu en paramètre :

```
$vlTable:=Table($1)
TABLEAU ALPHA(31;taChamps;Lire numero dernier champ($vlTable))
Boucle($vlChamp;Taille tableau(taChamps);1;-1)
   Si(Est un numero de champ valide($vlTable;$vlChamp))
      taChamps{$vlChamp}:=Nom du champ($vlTable;$vlChamp)
   Sinon
      SUPPRIMER DANS TABLEAU(taChamps;$vlChamp)
   Fin de si
Fin de boucle
```

Lire numero derniere table

Lire numero derniere table -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Numéro de table le plus élevé dans la base

Description

Lire numero derniere table retourne le numéro de table le plus élevé parmi les tables de la base.
Les tables sont numérotées dans l'ordre dans lequel elles ont été créées. Si aucune table n'a été supprimée dans la base, cette commande retourne donc le nombre de tables présentes dans la base. Dans le cadre de boucles itératives sur les numéros de tables de la base, vous devez utiliser la commande **Est un numero de table valide** afin de vérifier que la table n'a pas été supprimée.

Exemple

L'exemple suivant initialise les éléments du tableau tabTables. Ce tableau peut être utilisé comme liste déroulante (ou onglets, zone de défilement, etc.) pour afficher dans un formulaire la liste des tables de la base :

```
TABLEAU ALPHA(31;tabTables;Lire numero derniere table)
Si(Lire numero derniere table>0) `si la base contient bien des tables
   Boucle($vlTables;Taille tableau(tabTables);1;-1)
      Si(Est un numero de table valide($vlTables))
         tabTables{$vlTables}:=Nom de la table($vlTables)
      Sinon
         SUPPRIMER DANS TABLEAU(tabTables;$vlTables)
      Fin de si
   Fin de boucle
Fin de si
```

LIRE PROPRIETES CHAMP

LIRE PROPRIETES CHAMP (ptrChp | numTable {; numChamp}; champType {; champLong {; indexé {; unique {; invisible}}}})

Paramètre	Type		Description
ptrChp | numTable	Pointeur, Entier long	⇒	Pointeur de champ ou Numéro de table
numChamp	Entier long	⇒	Numéro de champ si un numéro de table est passé en premier paramètre
champType	Entier long	⇐	Type de champ
champLong	Entier long	⇐	Longueur du champ (si alphanumérique)
indexé	Booléen	⇐	Vrai = Indexé, Faux = Non indexé
unique	Booléen	⇐	Vrai = Unique, Faux = Non unique
invisible	Booléen	⇐	Vrai = Invisible, Faux = Visible

Description

La commande **LIRE PROPRIETES CHAMP** retourne des informations sur le champ désigné par *numTable* et *numChamp* ou par *ptrChp*.
Vous pouvez soit passer :

- les numéros de table et de champ dans *numTable* et *numChamp*
- ou un pointeur vers le champ dans *ptrChp*.

Après l'appel :

- Le paramètre *champType* retourne le type du champ. Le paramètre variable *champType* reçoit l'une des valeurs prédéfinies par les constantes de 4D (thème **Types champs et variables**) :

Constante	Type	Valeur
Est un BLOB	Entier long	30
Est un booléen	Entier long	6
Est un champ alpha	Entier long	0
Est un entier	Entier long	8
Est un entier 64 bits	Entier long	25
Est un entier long	Entier long	9
Est un float	Entier long	35
Est un numérique	Entier long	1
Est un objet	Entier long	38
Est un texte	Entier long	2
Est une date	Entier long	4
Est une heure	Entier long	11
Est une image	Entier long	3
Est une sous table	Entier long	7

- Le paramètre *champLong* retourne la longueur du champ si celui-ci est de type Alpha (ce qui signifie que vous obtenez *champType*=Est un champ alpha). La valeur de *champLong* n'est pas significative pour les autres types de champ.
- Le paramètre *indexé* retourne Vrai si le champ est indexé, Faux sinon. La valeur de *indexé* est significative pour les champs de type Alphanumérique, Entier, Entier long, Réel, Date, Heure et Booléen.
- Le paramètre *unique* retourne Vrai si le champ dispose de l'attribut "Unique", Faux sinon.
- Le paramètre *invisible* retourne Vrai si le champ dispose de l'attribut "Invisible", Faux sinon. L'attribut Invisible permet de masquer le champ dans les éditeurs standard de 4D (étiquettes, graphes...).

Exemple 1

Dans l'exemple suivant, les variables *vType*, *vLong*, *vIndex*, *vUnique* et *vInvisible* prennent pour valeur les propriétés du troisième champ de la première table :

```
LIRE PROPRIETES CHAMP(1;3;vType;vLong;vIndex;vUnique;vInvisible)
```

Exemple 2

L'exemple suivant récupère dans les variables *vType*, *vLong*, *vIndex*, *vUnique* et *vInvisible* les propriétés du champ [Table3]Champ2 :

```
LIRE PROPRIETES CHAMP(->[Table3]Champ2;vType;vLong;vIndex;vUnique;vInvisible)
```

LIRE PROPRIETES LIEN

LIRE PROPRIETES LIEN (ptrChp | numTable {; numChamp}; tableDest ; champDest {; discriminant {; allerAuto {; retourAuto}}})

Paramètre	Type		Description
ptrChp | numTable	Pointeur, Entier long	⇒	Pointeur de champ ou Numéro de table
numChamp	Entier long	⇒	Numéro de champ si un numéro de table est passé en premier paramètre
tableDest	Entier long	⇐	Numéro de la table cible ou 0 si aucun lien ne part du champ
champDest	Entier long	⇐	Numéro du champ cible ou 0 si aucun lien ne part du champ
discriminant	Entier long	⇐	Numéro du champ discriminant ou 0 si aucun champ discriminant
allerAuto	Booléen	⇐	Vrai = Lien aller automatique, Faux = Lien aller manuel
retourAuto	Booléen	⇐	Vrai = Lien retour automatique, Faux = Lien retour manuel

Description

La commande **LIRE PROPRIETES LIEN** retourne les propriétés du lien, s'il y en a un, qui part du champ source, désigné par *numTable* et *numChamp* ou par *ptrChp*.
Vous pouvez passer :

- soit des numéros de table et de champ dans *numTable* et *numChamp*,
- soit un pointeur vers le champ dans *ptrChp*.

Après l'exécution de la commande :

- Les paramètres *tableDest* et *champDest* contiennent respectivement le numéro de la table et du champ vers lesquels pointe le lien partant du champ source. Si aucun lien ne part du champ, ces paramètres contiennent 0.
- Le paramètre *discriminant* contient le numéro du champ discriminant (appartenant à la table cible) défini pour le lien. Si aucun champ discriminant n'a été défini pour le lien ou si aucun lien ne part du champ source, ce paramètre contient 0.
- Les paramètres *allerAuto* et *retourAuto* retournent Vrai si respectivement les options "Lien aller auto" et "Lien retour auto" ont été cochées pour le lien, Faux sinon.

Note : Les deux derniers paramètres retournent également Vrai si aucun lien ne part du champ source (dans ce cas, leur valeur est non significative). La valeur des paramètres *tableDest* et *champDest* vous permet de vous assurer de l'existence d'un lien.

LIRE PROPRIETES SAISIE CHAMP

LIRE PROPRIETES SAISIE CHAMP (ptrChp | numTable {; numChamp}; énumération ; obligatoire ; nonSaisissable ; nonModifiable)

Paramètre	Type		Description
ptrChp | numTable	Pointeur, Entier long	⇒	Pointeur de champ ou Numéro de table
numChamp	Entier long	⇒	Numéro de champ si un numéro de table est passé en premier paramètre
énumération	Chaîne	⇐	Nom de l'énumération associée ou Chaîne vide
obligatoire	Booléen	⇐	Vrai = Obligatoire, Faux = Facultatif
nonSaisissable	Booléen	⇐	Vrai = Non saisissable, Faux = Saisissable
nonModifiable	Booléen	⇐	Vrai = Non modifiable, Faux = Modifiable

Description

La commande **LIRE PROPRIETES SAISIE CHAMP** retourne les propriétés relatives à la saisie de données du champ désigné par *numTable* et *numChamp* ou par *ptrChp*.
Vous pouvez passer :

- soit des numéros de table et de champ dans *numTable* et *numChamp*,
- soit un pointeur vers le champ dans *ptrChp*.

Les propriétés retournées par cette commande sont celles qui ont été définies au niveau de la fenêtre de structure de la base. Des propriétés similaires peuvent également être définies au niveau des formulaires.

Après l'exécution de la commande :

- Le paramètre *énumération* contient le nom de l'énumération associée au champ, s'il y en a une. Il est possible d'associer un énumération aux champs de type Alpha, Texte, Numérique, Entier, Entier long, Date, Heure et Booléen.
  Si aucune énumération n'est associée au champ, ou si son type n'admet pas l'association d'énumération, une chaîne vide ("") est retournée.
- Le paramètre *obligatoire* retourne Vrai si le champ dispose de l'attribut "Obligatoire", Faux sinon. L'attribut "Obligatoire" peut être associé aux champs de tous types, hormis BLOB.
- Le paramètre *nonSaisissable* retourne Vrai si le champ dispose de l'attribut "Non saisissable", Faux sinon. Un champ non saisissable ne peut qu'être lu, il n'accepte aucune saisie de données. L'attribut "Non saisissable" peut être associé aux champs de tous types, hormis BLOB.
- Le paramètre *nonModifiable* retourne Vrai si le champ dispose de l'attribut "Non modifiable", Faux sinon. Un champ non modifiable n'accepte qu'une seule saisie, et ne peut plus être modifié par la suite. L'attribut "Non modifiable" peut être associé aux champs de tous types, hormis BLOB.

LIRE PROPRIETES TABLE

LIRE PROPRIETES TABLE (ptrTable | numTable ; invisible {; trigSvgdeNouv {; trigSvgdeEnr {; trigSupprEnr {; trigChargEnr}}}})

Paramètre	Type		Description
ptrTable | numTable	Pointeur, Entier long	⇒	Pointeur de table ou Numéro de table
invisible	Booléen	⇐	Vrai = Invisible, Faux = Visible
trigSvgdeNouv	Booléen	⇐	Vrai = Trigger “Sur sauvegarde nouvel enreg” activé, sinon Faux
trigSvgdeEnr	Booléen	⇐	Vrai = Trigger “Sur sauvegarde enregistrement” activé, sinon Faux
trigSupprEnr	Booléen	⇐	Vrai = Trigger “Sur suppression enreg” activé, sinon Faux
trigChargEnr	Booléen	⇐	*** Ne pas utiliser (obsolète) ***

Description

La commande **LIRE PROPRIETES TABLE** retourne les propriétés de la table désignée par *ptrTable* ou *numTable*. Vous pouvez passer dans le premier paramètre soit un pointeur vers la table, soit le numéro de la table.

Après l’exécution de la commande :

- Le paramètre *invisible* retourne Vrai si la table dispose de l’attribut “Invisible”, Faux sinon. L’attribut “Invisible” permet de masquer la table dans les éditeurs standard de 4D (étiquettes, graphes...).
- Le paramètre *trigSvgdeNouv* retourne Vrai si le trigger “Sur sauvegarde nouvel enreg” a été activé pour la table, Faux sinon.
- Le paramètre *trigSvgdeEnr* retourne Vrai si le trigger “Sur sauvegarde enregistrement” a été activé pour la table, Faux sinon.
- Le paramètre *trigSupprEnr* retourne Vrai si le trigger “Sur suppression enreg” a été activé pour la table, Faux sinon.

Nom de la table

Nom de la table (numTable | ptrTable) -> Résultat

Paramètre	Type		Description
numTable | ptrTable	Entier long, Pointeur	⇒	Numéro de table ou pointeur de table
Résultat	Chaîne	⇐	Nom de la table

Description

Nom de la table retourne le nom de la table dont le numéro ou le pointeur a été passé dans *numTable* ou *ptrTable*.

Exemple

La méthode suivante est un exemple de méthode générique qui affiche les enregistrements d'une table. La référence à la table est passée en tant que pointeur vers la table. La commande **Nom de la table** est utilisée pour inclure le nom de la table dans la barre de titre de la fenêtre :

```
  ` Méthode projet AFFICHER SELECTION COURANTE
  ` AFFICHER SELECTION COURANTE (Pointeur)
  ` AFFICHER SELECTION COURANTE (->[Table] )

CHANGER TITRE FENETRE(" Enregistrements pour "+Nom de la table($1)) ` Fixer le titre de la fenêtre
VISUALISER SELECTION($1->) ` Afficher la sélection
```

Nom du champ

Nom du champ (ptrChamp | numTable {; numChamp}) -> Résultat

Paramètre	Type		Description
ptrChamp | numTable	Pointeur, Entier long	➡	Pointeur vers un champ ou Numéro de table
numChamp	Entier long	➡	Numéro de champ si un numéro de table est passé en premier paramètre
Résultat	Chaîne	⮌	Nom du champ

Description

La commande **Nom du champ** retourne le nom du champ dont vous avez passé le pointeur dans *ptrChamp*, ou dont vous avez passé les numéros de table et de champ dans *numTable* et *numChamp*.

Exemple 1

L'exemple suivant assigne au second élément du tableau **ChampTableau{1}** (**ChampTableau** étant un tableau à deux dimensions) le nom du second champ de la première table :

```
ChampTableau{1}{2}:=Nom du champ(1;2)
```

Exemple 2

L'exemple suivant assigne au second élément du tableau **ChampTableau{1}** (**ChampTableau** étant un tableau à deux dimensions) le nom du champ *[MaTable]MonChamp* :

```
ChampTableau{1}{2}:=Nom du champ(->[MaTable]MonChamp)
```

Exemple 3

L'exemple suivant affiche une boîte de dialogue d'alerte. Nous passons à cette méthode un pointeur vers un champ :

```
ALERTE("Le numéro du champ "+Nom du champ($1)+" de la table "+Nom de la table(Table($1))+" doit faire plus de cinq caractères.")
```

REACTIVER INDEX

REACTIVER INDEX (laTable {; *})

Paramètre	Type		Description
laTable	Table	⇒	Table pour laquelle réactiver les index
*	Opérateur	⇒	Si passé = indexation asynchrone

Description

La commande **REACTIVER INDEX** réactive tous les index de *laTable* s'ils ont été préabalement suspendus via la commande **SUSPENDRE INDEX**. Si les index de *laTable* n'étaient pas suspendus, la commande ne fait rien.

Dans la plupart des cas, l'exécution de cette commande provoquera la reconstruction des index de *laTable*.
Si vous passez le paramètre optionnel *, la reconstruction des index sera effectuée en mode asynchrone. Ce mode signifie que la méthode appelant la commande poursuivra son exécution après cet appel, que l'indexation soit terminée ou non. Si vous omettez ce paramètre, la reconstruction des index bloquera l'exécution de la méthode jusqu'à ce que l'opération soit terminée.

La commande **REACTIVER INDEX** ne peut être appelée que depuis 4D Server ou un 4D local. Si cette commande est exécutée depuis un poste 4D distant, l'erreur -10513 est générée. Cette erreur peut être interceptée par une méthode installée par la commande **APPELER SUR ERREUR**.

RECHARGER DONNEES EXTERNES

RECHARGER DONNEES EXTERNES (leChamp)

Paramètre	Type		Description
leChamp	Texte, BLOB, Image, Objet	⇨	Champ pour lequel recharger les données

Description

La commande **RECHARGER DONNEES EXTERNES** vous permet de recharger en mémoire le contenu d'un fichier de stockage externe associé à un champ de type Blob, Image, Texte ou Objet.

Cette commande est utile dans le cas où le champ d'un enregistrement déjà chargé en mémoire est modifié sur le disque par une autre application (les fichiers de stockage externe des champs sont toujours accessibles en écriture). Par exemple, une image utilisée dans un champ image est modifiée par un éditeur graphique puis sauvegardée sur disque.

Il est alors nécessaire de demander le rechargement des données à l'aide de la commande **RECHARGER DONNEES EXTERNES** pour mettre à jour le contenu du champ s'il est affiché dans un formulaire.

Note : La commande **RECHARGER DONNEES EXTERNES** fonctionne uniquement sur 4D local ou 4D Server. Il n'est pas possible de recharger individuellement un champ avec 4D en mode distant. Il est nécessaire dans ce contexte de recharger l'ensemble de l'enregistrement (à l'aide de la commande **CHARGER ENREGISTREMENT** par exemple).

REGENERER TABLE MANQUANTE

REGENERER TABLE MANQUANTE (nomTable)

Paramètre	Type		Description
nomTable	Texte	⇨	Nom de table manquante à regénérer

Description

La commande **REGENERER TABLE MANQUANTE** reconstruit la table manquante dont vous avez passé le nom dans le paramètre *nomTable*. Lorsqu'une table manquante est reconstruite, elle devient visible dans l'éditeur de Structure et ses données sont de nouveau accessibles.

Les tables manquantes sont des tables dont les données sont présentes dans le fichier de données mais qui n'existent pas au niveau de la structure. Vous pouvez identifier les tables manquantes éventuellement présentes dans l'application à l'aide de la commande **LIRE NOMS TABLES MANQUANTES**.

Si la table désignée par le paramètre *nomTable* n'est pas une table manquante de la base, la commande ne fait rien.

Exemple

Cette méthode regénère toutes les tables manquantes éventuellement présentes dans la base :

```
TABLEAU TEXTE($tMissingTables;0)
LIRE NOMS TABLES MANQUANTES($tMissingTables)
$SizeArray:=Taille tableau($tMissingTables)
Si($SizeArray#0)
 // Remplir le tableau avec les noms de toutes les tables de la base
   TABLEAU TEXTE(tabTables;Lire numero derniere table)
   Si(Lire numero derniere table>0)    //S'il y a bien des tables
      Boucle($vlTables;Taille tableau(tabTables);1;-1)
         Si(Est un numero de table valide($vlTables))
            tabTables{$vlTables}:=Nom de la table($vlTables)
         Sinon
            SUPPRIMER DANS TABLEAU(tabTables;$vlTables)
         Fin de si
      Fin de boucle
   Fin de si
   Boucle($i;1;$SizeArray)
      Si(Chercher dans tableau(tabTables;$tMissingTables{$i})=-1)
         CONFIRMER("Regénérer la table"+$tMissingTables{$i}+" ?")
         Si(OK=1)
            REGENERER TABLE MANQUANTE($tMissingTables{$i})
         Fin de si
      Sinon
         ALERTE("Impossible de régénérer la table "+$tMissingTables{$i}+" car il y a déjà une table de ce nom dans
la base.")
      Fin de si
   Fin de boucle
 Sinon
   ALERTE("Pas de tables à regénérer.")
 Fin de si
```

SUPPRIMER INDEX

SUPPRIMER INDEX (ptrChp | nomIndex {; *})

Paramètre	Type		Description
ptrChp | nomIndex	Pointeur, Chaîne	⇒	Pointeur vers le champ duquel supprimer les index ou Nom de l'index à supprimer
*	Opérateur	⇒	Si passé = opération asynchrone

Description

La commande **SUPPRIMER INDEX** permet de supprimer un ou plusieurs index existant dans la base. Vous pouvez passer en paramètre soit un pointeur vers un champ, soit un nom d'index :

- Si vous passez un pointeur vers un champ (*ptrChp*), tous les index associés au champ seront supprimés. Il peut s'agir d'index de mots-clés ou d'index standard. En revanche, si le champ est inclus dans un ou plusieurs index composite(s), ils ne sont pas supprimés (vous devez passer un nom d'index).
- Si vous passez nom d'index (*nomIndex*), seul l'index désigné sera supprimé. Il peut s'agir d'index de mots-clés, d'index standard ou d'index composites.

Le paramètre facultatif *, lorsqu'il est passé, permet d'effectuer la désindexation en mode asynchrone. Dans ce mode, la méthode d'origine poursuit son exécution après l'appel de la commande, que la suppression d'index soit terminée ou non.

S'il n'existe pas d'index correspondant à *ptrChp* ou à *nomIndex*, la commande ne fait rien.

Exemple

Cet exemple illustre les deux syntaxes de la commande :

```
  `Suppression de tous les index liés au champ Nom
SUPPRIMER INDEX(->[Clients]Nom)
  `Suppression de l'index nommé "CPVille"
SUPPRIMER INDEX("CPVille")
```

SUSPENDRE INDEX

SUSPENDRE INDEX (laTable)

Paramètre	Type		Description
laTable	Table	⇨	Table pour laquelle suspendre les index

Description

La commande **SUSPENDRE INDEX** désactive temporairement tous les index de *laTable*, hormis l'index de la clé primaire.

Les index ne sont pas physiquement supprimés des données (fichier .4DIndx) et de la structure de la base (_USER_INDEXES, cf. **Tables système**), ils sont rendus invalides et par conséquent ne sont plus mis à jour. Lorsque les index sont désactivés, toutes les opérations effectuées sur *laTable* (recherches, tris, ajouts, modifications et suppressions d'enregistrements) n'utilisent plus les index.

Cette commande est principalement utile dans le contexte de l'importation ou la modification massive de données dans des tables comportant plusieurs index. 4D devant mettre à jour les index à chaque validation d'enregistrement, l'opération peut prendre beaucoup de temps. Désactiver les index au préalable permet d'accélérer significativement l'opération.

Pour réactiver les index à l'issue de l'opération, il suffit d'appeler la commande **REACTIVER INDEX** sur *laTable*.

Note : Il est possible d'obtenir un résultat similaire en utilisant les commandes **CREER INDEX** et **SUPPRIMER INDEX**, mais il est nécessaire d'appeler ces commandes pour chaque index de *laTable*.

Si vous appelez la commande **SUSPENDRE INDEX** sur une table puis quittez la base sans que la commande **REACTIVER INDEX** ait été exécutée sur cette table, tous les index de la table seront automatiquement reconstruits au prochain démarrage de la base.

Note : Cette commande ne peut pas être exécutée depuis un 4D distant.

Exemple

Exemple de méthode d'import massif de données :

```
SUSPENDRE INDEX([Articles])
IMPORTER DONNEES("GrosImport.txt") //Importation
REACTIVER INDEX([Articles])
```

Table

Table (numTable | unPtr) -> Résultat

Paramètre	Type		Description
numTable | unPtr	Entier long, Pointeur	⇨	Numéro de table ou Pointeur de table ou Pointeur de champ
Résultat	Entier long, Pointeur	⇦	Pointeur de table si un Numéro de table est passé, Numéro de table si un Pointeur de table est passé, Numéro de table si un Pointeur de champ est passé

Description

Table a trois syntaxes différentes.

- Si vous passez un numéro de table dans *numTable*, **Table** retourne un pointeur sur la table.
- Si vous passez un pointeur de table dans *unPtr*, **Table** retourne le numéro de la table.
- Si vous passez un pointeur de champ dans *unPtr*, **Table** retourne le numéro de table du champ.

Exemple 1

Dans cet exemple, la variable *ptrTable* reçoit un pointeur sur la table n°3 :

```
ptrTable:=Table(3)
```

Exemple 2

Si vous passez *ptrTable* à la fonction **Table**, elle retourne 3. Par exemple, dans la ligne suivante, la variable *numTable* prend la valeur 3 :

```
numTable:=Table(ptrTable)
```

Exemple 3

Dans l'exemple suivant, la variable *numTable* est égale au numéro de la table [Table3] :

```
numTable:=Table(->[Table3])
```

Exemple 4

Dans l'exemple suivant, la variable *numTable* est égale au numéro de la table à laquelle appartient le champ [Table3]Champ1 :

```
numTable:=Table(->[Table3]Champ1)
```

Documents système

- Présentation des documents système
- Ajouter a document
- ASSOCIER TYPES FICHIER
- CHANGER CREATEUR DOCUMENT
- CHANGER POSITION DANS DOCUMENT
- CHANGER PROPRIETES DOCUMENT
- CHANGER TAILLE DOCUMENT
- CHANGER TYPE DOCUMENT
- Convertir chemin POSIX vers systeme
- Convertir chemin systeme vers POSIX
- COPIER DOCUMENT
- Createur document
- CREER ALIAS
- Creer document
- CREER DOSSIER
- DEPLACER DOCUMENT
- Document vers texte
- FERMER DOCUMENT
- Lire chemin document localise
- LIRE ICONE DOCUMENT
- LISTE DES DOCUMENTS
- LISTE DES DOSSIERS
- LISTE DES VOLUMES
- MONTRER SUR DISQUE
- Ouvrir document
- Position dans document
- PROPRIETES DOCUMENT
- PROPRIETES DU VOLUME
- RESOUDRE ALIAS
- Selectionner document
- Selectionner dossier
- SUPPRIMER DOCUMENT
- SUPPRIMER DOSSIER
- Taille document
- Tester chemin acces
- TEXTE VERS DOCUMENT
- Type document

Présentation des documents système

Introduction

Tous les documents et applications que vous utilisez sur votre ordinateur sont stockés en tant que **fichiers** sur le ou les disques durs **connectés** ou **montés** sur votre ordinateur, ou encore sur des disquettes et autres supports de stockage permanent. Dans cette documentation ainsi que dans 4D, les termes **fichier** ou **document** sont indifféremment employés pour désigner ces documents et applications. Cependant, la plupart des commandes de ce thème utilisent le mot document car, généralement, vous les utiliserez pour accéder à des documents (par opposition à des fichiers d'application ou des fichiers système) sur disque.

Un disque dur peut être formaté de manière à comporter une ou plusieurs partitions. Chaque partition s'appelle un **volume**. Peu importe que ces volumes soient des partitions physiquement présentes sur le même disque dur ou non, au niveau de 4D, ces volumes sont considérés comme des entités séparées et équivalentes.

Un volume peut être situé sur un disque dur physiquement connecté à votre machine ou monté par le réseau par l'intermédiaire d'un protocole de partage de fichiers tel que TCP/IP, AFP ou SMB (Macintosh). Quel que soit le cas, au niveau de 4D, ces volumes sont considérés de la même façon lorsque vous utilisez les commandes du thème Documents Système (à moins que vous n'en décidiez autrement et utilisiez des plug-ins 4D pour étendre les capacités de votre application dans ce domaine).

Chaque volume a un **nom de volume**. Sous Windows, les volumes sont désignés par une lettre suivie de deux points. Généralement, **C:** et **D:** désignent les volumes que vous utilisez pour lancer votre système (à moins que vous n'ayez configuré votre PC différemment). Ensuite, les lettres **E:** à **Z:** sont utilisées pour les volumes supplémentaires connectés à votre PC (lecteurs DVD, autres lecteurs, lecteurs réseau, etc.). Sous Mac OS, les volumes ont des noms communs (ces noms sont ceux que vous visualisez sur le bureau au niveau du Finder).

Normalement, vous classez vos documents dans des **dossiers** qui peuvent eux-mêmes contenir d'autres dossiers. Il n'est pas conseillé d'accumuler des centaines ou des milliers de fichiers au même niveau d'un volume. C'est un fouillis, qui de plus qui ralentit votre système. Sous Windows, un dossier est parfois encore appelé un “répertoire”.

Pour identifier un document de manière certaine, vous avez besoin de connaître le nom du volume, le nom du ou des dossiers(s) dans le(s)quel(s) se trouve le document, ainsi que le nom du document lui-même. Si vous concaténez tous ces noms, vous obtenez le **chemin d'accès** à ce document. Dans le nom de ce chemin, les noms de dossiers sont séparés par un caractère spécial appelé **séparateur de dossier**. Sous Windows, ce caractère est la barre oblique inversée \, sous Mac OS ce sont les deux-points *:*

Examinons un exemple. Vous disposez d'un document **Important** situé dans le dossier **Mémos**, lui-même situé dans le dossier **Documents**, lui-même situé dans le dossier **EnCours**.

Si, sous Windows, l'ensemble est situé sur le volume **C:** , le chemin d'accès au document est donc :
C:\EnCours\Documents\Mémos\Important.TXT

Note : Le caractère \ est également utilisé par l'éditeur de méthodes de 4D pour désigner des séquences d'échappement. Pour éviter tout problème d'interprétation, l'éditeur transforme automatiquement les chemins d'accès du type **C:\Disque** en **C:\\Disque**. Pour plus d'informations, reportez-vous ci-dessous au paragraphe “Spécification des noms et chemins d'accès des documents”.

Si, sous Mac OS, l'ensemble est situé sur le volume **Interne**, le chemin d'accès au document est donc :
Interne:EnCours:Documents:Mémos:Important

Notez que, sous Windows, avec cet exemple, le nom du document contient le suffixe .TXT. Nous verrons pourquoi plus loin dans cette section.

Quelle que soit la plate-forme, le chemin d'accès à un document peut être exprimé sous la forme suivante : **VolNom DosSep { DosNom DosSep { DosNom DosSep { ... } } } DocNom**.

Tous les documents (fichiers) situés sur des volumes ont plusieurs caractéristiques généralement appelées **attributs** ou **propriétés** : par exemple le **nom** du document lui-même.

RefDoc : numéro de référence de document

Un document est **ouvert en mode lecture/écriture**, **ouvert en mode lecture** ou **fermé**. Avec les commandes 4D, un document ne peut être ouvert en mode lecture/écriture que par un process à la fois. Un process peut ouvrir plusieurs documents, plusieurs process peuvent ouvrir de multiples documents, vous pouvez ouvrir un même document en mode lecture autant de fois que nécessaire, mais vous ne pouvez pas ouvrir le même document en mode lecture/écriture deux fois en même temps.

Vous ouvrez un document à l'aide des fonctions **Ouvrir document**, **Creer document** et **Ajouter a document**. Les fonctions **Creer document** et **Ajouter a document** ouvrent automatiquement les documents en mode lecture/écriture. Seule la fonction **Ouvrir document** permet de choisir le mode d'ouverture. Une fois que le document est ouvert en lecture/écriture, vous pouvez lire et écrire des caractères dans ce document (cf. les commandes **RECEVOIR PAQUET** et **ENVOYER PAQUET**). Lorsque vous en avez terminé avec un document, il est préférable de le fermer — avec la commande **FERMER DOCUMENT**.

Tous les documents ouverts sont désignés au moyen de l'expression *RefDoc*, retournée par les commandes **Ouvrir document**, **Creer document** et **Ajouter a document**. Une *RefDoc* identifie de façon unique un document ouvert. C'est une expression de type Heure. Toutes les commandes fonctionnant avec des documents ouverts attendent une *RefDoc* comme paramètre. Si vous passez une *RefDoc* incorrecte à l'une de ces commandes, une erreur du gestionnaire de fichiers est générée.

Note : Lorsqu'elle est appelée depuis un process préemptif, une référence *RefDoc* est utilisable uniquement depuis ce process préemptif. Lorsqu'elle est appelée depuis un process coopératif, une référence *RefDoc* est utilisable depuis n'importe quel autre process coopératif.

Gestion des erreurs E/S

Quand vous accédez à des documents (ouverture, fermeture, suppression, changement de nom, copie), quand vous modifiez les propriétés d'un document ou quand vous lisez et écrivez des caractères dans un document, des erreurs d'entrée/sortie (E/S) peuvent se produire. Un document peut ne pas avoir été trouvé ; il peut être verrouillé ; il peut être déjà ouvert en écriture. Vous pouvez repérer ces erreurs grâce à une méthode de gestion des erreurs installée par la commande **APPELER SUR ERREUR**. La plupart des erreurs qui peuvent se produire lors de l'utilisation des commandes du thème documents système est décrite dans la section **Erreurs du gestionnaire de fichiers du système (-124 -> -33)**.

La variable système Document

Les commandes **Ouvrir document**, **Creer document**, **Ajouter a document** et **Selectionner document** vous permettent d'accéder à un document par

les boîtes de dialogue standard d'ouverture ou d'enregistrement de fichiers. Quand vous accédez à un document par ces boîtes de dialogue standard, 4D retourne le chemin d'accès complet du document dans la variable système *Document*. Ne confondez pas cette variable système avec le paramètre *document* qui apparaît dans la liste des paramètres des commandes.

Spécification des noms et chemins d'accès des documents

La plupart des routines de cette section attendant un nom de document acceptent à la fois un nom et un chemin d'accès au document (*). Si vous passez un nom, la commande cherche le document dans le dossier de la base. Si vous passez un chemin d'accès, il doit être valide.

Si vous passez un nom ou un chemin d'accès incorrect, la commande génère une erreur du gestionnaire de fichiers que vous pouvez intercepter avec une méthode d'**APPELER SUR ERREUR**.

(*) sauf cas contraire spécifié explicitement.

Saisie de chemins d'accès Windows et séquences d'échappement

L'éditeur de méthodes de 4D permet d'utiliser des séquences d'échappement. Une séquence d'échappement est une suite de caractères permettant de remplacer un caractère "spécial". La séquence débute par le caractère barre oblique inversée (antislash) \, suivi d'un caractère. Par exemple, \t est une séquence d'échappement pour le caractère Tabulation.
Le caractère \ est aussi utilisé comme séparateur dans les chemins d'accès sous Windows. En général, 4D interprétera correctement les chemins d'accès Windows saisis dans l'éditeur de méthodes en remplaçant automatiquement les barres simples \ par des doubles barres \\. Par exemple, **C:\Dossier** deviendra **C:\\Dossier**.
Toutefois, si vous écrivez **C:\MesDocuments\Nouveaux**, 4D affichera **C:\\MesDocuments\Nouveaux**. Dans ce cas, le second \ est incorrectement interprété \N (séquence d'échappement existante). Vous devez donc saisir un double \\ lorsque vous souhaitez insérer une barre oblique inversée devant un caractère utilisé dans une des séquences d'échappement reconnues par 4D.

Les séquences d'échappement reconnues par 4D sont les suivantes :

Séquence d'échappement	Caractère remplacé
\n	LF (Retour ligne)
\t	HT (Tabulation)
\r	CR (Retour chariot)
\\	\ (Barre oblique inversée)
\"	" (Guillemets)

Chemin d'accès absolu ou relatif

La plupart des commandes 4D de gestion des documents et des dossiers acceptent des chemins d'accès relatifs ou absolus :

- Les **chemins d'accès relatifs** définissent un emplacement par rapport à un dossier présent sur le disque. Dans 4D, les chemins d'accès relatifs sont généralement exprimés par rapport au dossier de la base, c'est-à-dire au dossier contenant le fichier de structure. Les chemins d'accès relatifs sont particulièrement utiles pour le déploiement d'applications en environnement hétérogène.
- Les **chemins d'accès absolus** définissent un emplacement à partir de la racine d'un volume. Ils ne dépendent pas de la position courante du dossier de la base.

Pour déterminer si un chemin d'accès passé à une commande doit être interprété en tant que chemin relatif ou absolu, 4D applique un algorithme spécifique pour chaque plate-forme.

Sous Windows

Si le paramètre contient seulement deux caractères **et** si le deuxième est un ':',
ou si le texte contient ':' et '\' comme deuxième et troisième caractère,
ou si le texte débute par "\\",
alors le chemin d'accès est **absolu**.

Dans tous les autres cas, le chemin d'accès est **relatif**.

Exemples avec la commande **CREER DOSSIER** :

```
CREER DOSSIER("lundi")  // chemin relatif
CREER DOSSIER("\lundi")  // chemin relatif
CREER DOSSIER("\lundi\mardi")  // chemin relatif
CREER DOSSIER("c:") // chemin absolu
CREER DOSSIER("d:\lundi") // chemin absolu
CREER DOSSIER("\\srv-Interne\tempo") // chemin absolu
```

Sous Mac OS

Si le texte débute par un séparateur de dossier ':',
ou s'il n'en contient aucun,
alors le chemin est **relatif**.

Dans tous les autres cas, il est **absolu**.

Exemples avec la commande **CREER DOSSIER** :

```
CREER DOSSIER("lundi") // chemin relatif
CREER DOSSIER("macintosh hd:") // chemin absolu
CREER DOSSIER("lundi:mardi") // chemin absolu (un volume doit s'appeler lundi)
CREER DOSSIER(":lundi:mardi") // chemin relatif
```

Méthodes projet utiles pour la gestion des documents sur disque

- **Détecter sur quelle plate-forme vous opérez**

Bien que 4D fournisse des commandes telles que **ASSOCIER TYPES FICHIER** destinées à éliminer les modifications de code liées aux particularités des plates-formes, lorsque vous commencez à travailler à un plus bas niveau en manipulant des documents sur disque, par exemple lorsque vous obtenez les chemins d'accès par programmation, vous avez besoin de savoir si vous fonctionnez sous Windows ou Mac OS.

La méthode projet ***Sous Windows*** listée ci-dessous vous permet de savoir si votre base tourne sous Windows :

```
// Méthode projet Sous Windows
// Sous Windows -> Booléen
// Sous Windows -> Vrai si la base est sous Windows

C_BOOLEEN($0)
C_ENTIER LONG($vlPlatform;$vlSystem;$vlMachine)

PROPRIETES PLATE FORME($vlPlatform;$vlSystem;$vlMachine)
$0:=($vlPlatform=Windows)
```

- **Extraire le nom de fichier d'un chemin d'accès complet (ou "nom long")**

Une fois que vous avez récupéré le "nom long" d'un fichier (c'est-à-dire le chemin d'accès+le nom du fichier), vous pouvez avoir besoin d'en extraire le nom du fichier seul, par exemple pour l'afficher comme titre d'une fenêtre. La méthode projet ***Nom long vers nom de fichier*** vous le permet, sous Windows et Mac OS.

```
// Méthode projet Nom long vers nom de fichier
// Nom long vers nom de fichier ( Chaîne ) -> Chaîne
// Nom long vers nom de fichier ( nom long ) -> nom de fichier

C_TEXTE($1;$0)
C_ENTIER LONG($viLen;$viPos;$viChar;$viDirSymbol)

$viDirSymbol:=Code de caractere(Séparateur dossier)
$viLen:=Longueur($1)
$viPos:=0
Boucle($viChar;$viLen;1;-1)
   Si(Code de caractere($1≤$viChar≥)=$viDirSymbol)
      $viPos:=$viChar
      $viChar:=0
   Fin de si
Fin de boucle
Si($viPos>0)
   $0:=Sous chaine($1;$viPos+1)
Sinon
   $0:=$1
Fin de si
Si(<>vbDebugOn) // Mettre la variable à Vrai ou Faux dans la méthode base Sur ouverture
   Si($0="")
      TRACE
   Fin de si
Fin de si
```

- **Extraire le chemin d'accès seul du chemin d'accès complet (ou "nom long")**

Une fois que vous avez récupéré le "nom long" d'un fichier (c'est-à-dire le chemin d'accès+le nom du fichier), vous pouvez avoir besoin d'en extraire uniquement le chemin d'accès au fichier, par exemple pour sauvegarder d'autres documents au même endroit. La méthode projet ***Nom long vers chemin accès*** vous le permet, sous Windows et Mac OS.

```
// Méthode projet Nom long vers chemin accès
// Nom long vers chemin accès ( Chaîne ) -> Chaîne
// Nom long vers chemin accès ( nom long ) -> chemin d'accès

C_TEXTE($1;$0)
C_ENTIER LONG($viLen;$viPos;$viChar;$viDirSymbol)

$viDirSymbol:=Code de caractere(Séparateur dossier)
$viLen:=Longueur($1)
$viPos:=0
Boucle($viChar;$viLen;1;-1)
   Si(Code de caractere($1≤$viChar≥)=$viDirSymbol)
      $viPos:=$viChar
      $viChar:=0
   Fin de si
Fin de boucle
Si($viPos>0)
   $0:=Sous chaine($1;1;$viPos)
Sinon
   $0:=$1
Fin de si
Si(<>vbDebugOn) // Mettre la variable à Vrai ou Faux dans la méthode base Sur ouverture
   Si($0="")
      TRACE
   Fin de si
Fin de si
```

Ajouter a document

Ajouter a document (nomFichier {; typeFichier}) -> Résultat

Paramètre	Type		Description
nomFichier	Chaîne	⇨	Nom du document ou Chemin d'accès complet au document ou Chaîne vide pour afficher la boîte de dialogue standard d'ouverture de fichiers
typeFichier	Chaîne	⇨	Liste des types de documents à filtrer, ou "*" pour ne pas filtrer les documents
Résultat	RefDoc	⇦	Numéro de référence du document

Description

La commande **Ajouter a document** "fait la même chose" que la commande **Ouvrir document** : elle vous permet d'ouvrir un document sur disque et de se placer à la fin du document.
La seule différence est que **Ajouter a document** se place initialement à la fin du document, alors que **Ouvrir document** se place au début.
Pour plus d'informations, reportez-vous à la description de la commande **Ouvrir document**.

Exemple

L'exemple suivant ouvre un document existant qui s'appelle “Note”, ajoute à la fin du document la chaîne “ et à bientôt” suivie d'un retour chariot puis le referme. Si le document contenait déjà la chaîne “Au revoir”, il contiendra la chaîne “Au revoir et à bientôt” suivie d'un retour chariot :

```
C_HEURE(vDoc)
vDoc:=Ajouter a document("Note.txt") ` Ouvrir le document Note
ENVOYER PAQUET(vDoc;" et à bientôt"+Caractere(13)) ` Ajouter la chaîne
FERMER DOCUMENT(vDoc) ` Fermer le document
```

ASSOCIER TYPES FICHIER

ASSOCIER TYPES FICHIER (macOS ; windows ; contexte)

Paramètre	Type		Description
macOS	Chaîne	⇒	Type de fichier Mac OS (4 caractères)
windows	Chaîne	⇒	Extension de fichier Windows
contexte	Chaîne	⇒	Chaîne affichée dans la liste déroulante des types de fichiers dans la boîte de dialogue d'ouverture de fichiers sous Windows

Description

ASSOCIER TYPES FICHIER vous permet d'associer une extension de fichier Windows à un type de fichier Mac OS.

Il suffit d'appeler cette routine une fois seulement pour associer des types de fichier dans une session de travail entière avec une base. Une fois que vous l'avez appelée, les commandes **Creer document**, **Ajouter a document**, et **Ouvrir fichier ressources**, lorsqu'elle sont exécutées sous Windows, vont automatiquement substituer l'extension de fichier Windows au type de fichier Mac OS que vous avez passé en paramètre à cette routine.

Dans le paramètre *macOS*, passez un type de fichier Macintosh de 4 caractères. Si vous ne passez pas une chaîne valide représentant un type de fichier, la commande ne fait rien et génère une erreur.

Dans le paramètre *windows*, passez une extension de fichier Windows de 1 à N caractères. Si vous ne passez pas une chaîne valide représentant une extension de fichier, la commande ne fait rien et génère une erreur.

Dans le paramètre *contexte*, passez la chaîne qui sera affichée dans le menu déroulant des types de fichiers dans la boîte de dialogue d'ouverture de fichiers sous Windows. La chaîne *contexte* est limitée à 32 caractères ; tout caractère supplémentaire est ignoré.

IMPORTANT: Une fois que vous avez associé une extension de fichier Windows à un type de fichier Mac OS, vous ne pouvez pas modifier ou supprimer cette association dans la même session de travail. Si vous avez besoin de modifier l'association pendant le développement d'une application 4D, il faut réouvrir la base et exécuter de nouveau la commande.

Exemple

La ligne de code suivante (elle peut faire partie de la **Méthode base Sur ouverture**) associe les fichiers de type MS-Word (sous Mac OS *"WDBN"*) à l'extension de fichier Windows *".DOC"*:

```
ASSOCIER TYPES FICHIER("WDBN";"DOC";"Documents Word")
```

Une fois cette ligne exécutée, le code suivant n'affiche que des documents Word dans la boîte de dialogue d'ouverture de fichiers sous Windows et Mac OS :

```
$DocRéf:=Ouvrir document("";"WDBN")
Si(OK=1)
   ` ...
Fin de si
```

CHANGER CREATEUR DOCUMENT

CHANGER CREATEUR DOCUMENT (nomFichier ; créateur)

Paramètre	Type		Description
nomFichier	Chaîne	⇨	Nom de document ou Chemin d'accès complet à un document
créateur	Chaîne	⇨	Créateur de fichier (Mac OS) ou Chaîne vide (Windows)

Description

La commande **CHANGER CREATEUR DOCUMENT** définit le créateur du document dont vous avez passé le nom ou le chemin d'accès complet dans *document*.

Vous passez le nouveau créateur du document dans *créateur*.

Sous Windows, cette commande ne fait rien.

Reportez-vous à la description du créateur de document dans la section **Présentation des documents système**.

CHANGER POSITION DANS DOCUMENT

CHANGER POSITION DANS DOCUMENT (docRef ; offset {; ancre})

Paramètre	Type		Description
docRef	RefDoc	⇒	Numéro de référence de document
offset	Réel	⇒	Position dans fichier (exprimée en octets)
ancre	Entier long	⇒	1 = Par rapport au début du fichier 2 = Par rapport à la fin du fichier 3 = Par rapport à la position courante

Description

Cette commande ne fonctionne qu'avec un document déjà ouvert, dont vous avez passé le numéro de référence dans le paramètre *docRef*.

CHANGER POSITION DANS DOCUMENT définit la position que vous passez dans *offset* comme étant celle à laquelle la prochaine lecture (**RECEVOIR PAQUET**) ou écriture (**ENVOYER PAQUET**) aura lieu.

Si vous omettez le paramètre optionnel *ancre*, la position est définie par rapport au début du document. Sinon, vous pouvez passer dans le paramètre *ancre* une des valeurs listées ci-dessus.

En fonction de l'*ancre* définie, vous pouvez passer des valeurs positives ou négatives dans le paramètre *offset*.

CHANGER PROPRIETES DOCUMENT

CHANGER PROPRIETES DOCUMENT (nomFichier ; verrouillé ; invisible ; créé le ; créé à ; modifié le ; modifié à)

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Nom du document ou Chemin d'accès complet au document
verrouillé	Booléen	⇒	Verrouillé (Vrai) ou non verrouillé (Faux)
invisible	Booléen	⇒	Invisible (Vrai) ou visible (Faux)
créé le	Date	⇒	Date de création
créé à	Heure	⇒	Heure de création
modifié le	Date	⇒	Date de dernière modification
modifié à	Heure	⇒	Heure de dernière modification

Description

La commande **CHANGER PROPRIETES DOCUMENT** modifie certaines informations du document dont vous avez passé le nom ou le chemin d'accès dans *document*.

Avant l'appel :

- Passez Vrai dans *verrouillé* pour verrouiller le document. Un document verrouillé ne peut être modifié. Passez Faux dans *verrouillé* pour déverrouiller un document.
- Passez Vrai dans *invisible* pour cacher le document. Passez Faux dans *invisible* pour rendre le document visible dans les fenêtres du bureau.
- Passez la date et l'heure de création du document dans *créé le* et *créé à*.
- Passez la date et l'heure de la dernière modification du document dans *modifié le* et *modifié à*.

L'heure et la date de création et de dernière modification sont gérées par le gestionnaire de fichiers de votre système, à chaque fois que vous créez ou modifiez un document. Cette commande vous permet de modifier ces propriétés, dans des buts particuliers. Reportez-vous à l'exemple de la commande **PROPRIETES DOCUMENT**.

CHANGER TAILLE DOCUMENT

CHANGER TAILLE DOCUMENT (docRef ; taille)

Paramètre	Type		Description
docRef	RefDoc	⇨	Numéro de référence de document
taille	Réel	⇨	Nouvelle taille (en octets) de document

Description

La commande **CHANGER TAILLE DOCUMENT** fixe la taille d'un document au nombre d'octets que vous avez passé dans *taille*.
Le document doit avoir été ouvert au préalable. Vous passez son numéro de référence dans *docRef*.
Sous Mac OS, c'est la taille de la data fork du document qui est modifiée.

CHANGER TYPE DOCUMENT

CHANGER TYPE DOCUMENT (nomFichier ; typeFichier)

Paramètre	Type		Description
nomFichier	Chaîne	⇨	Nom de document ou Chemin d'accès complet à un document
typeFichier	Chaîne	⇨	Extension de fichier Windows ou type de fichier Mac OS (chaîne de 4 caractères)

Description

La commande **CHANGER TYPE DOCUMENT** définit le type du document dont vous avez passé le nom ou le chemin d'accès complet dans *nomFichier*.

Vous passez le nouveau type du document dans *typeFichier*. Reportez-vous à la description des types de documents dans les sections **Présentation des documents système** et **Type document**.

Sous Windows, la commande modifie l'extension et donc le nom du fichier décrit par *nomFichier*. Par exemple, l'instruction **CHANGER TYPE DOCUMENT**("C:\\Docs\\Facture.asc";"TEXT") renomme le fichier "Facture.asc" en "Facture.txt" (dans 4D, le type Mac "TEXT" est associé au type Windows "txt").

Si le type n'est pas associé dans 4D, il suffit de passer directement les trois lettres de l'extension. Par exemple, l'instruction **CHANGER TYPE DOCUMENT**("C:\\Docs\\Facture.asc";"zip") renomme le fichier "Facture.asc" en "Facture.zip".

Convertir chemin POSIX vers systeme

Convertir chemin POSIX vers systeme (cheminPosix {; *}) -> Résultat

Paramètre	Type		Description
cheminPosix	Texte	➡	Chemin d'accès POSIX
*	Opérateur	➡	Option d'encodage
Résultat	Texte	⮌	Chemin d'accès exprimé en syntaxe système

Description

La commande **Convertir chemin POSIX vers systeme** convertit un chemin d'accès exprimé avec la syntaxe POSIX (Unix) en chemin d'accès exprimé avec la syntaxe système.

Passez dans le paramètre *cheminPosix* le chemin d'accès complet à un fichier ou un dossier, exprimé avec la syntaxe POSIX. Ce chemin doit être absolu (il doit débuter par le caractère "/"). Vous devez passer un chemin disque, il n'est pas possible de passer de chemin réseau (débutant par exemple par ftp://ftp.monsite.fr).

La commande retourne le chemin d'accès complet du fichier ou du dossier exprimé dans la syntaxe du système courant.

Le paramètre optionnel * vous permet d'indiquer si le paramètre *cheminPosix* est encodé. Si c'est le cas, vous devez passer ce paramètre sinon la conversion sera invalide. La commande retourne le chemin d'accès sans encodage.

Exemple 1

Exemple sous Mac OS :

```
$chemin:=Convertir chemin POSIX vers systeme("/Volumes/machd/file 2.txt") //retourne "machd:file 2.txt"
$chemin:=Convertir chemin POSIX vers systeme("/Volumes/machd/file%202.txt";*)  //retourne "machd:file 2.txt"
$chemin:=Convertir chemin POSIX vers systeme("/file 2.txt") //retourne "machd:file 2.txt" si machd est le disque de démarrage
```

Exemple 2

Exemples sous Windows :

```
$chemin:=Convertir chemin POSIX vers systeme("c:/docs/file 2.txt") //retourne "c:\\docs\\file 2.txt"
$chemin:=Convertir chemin POSIX vers systeme("c:/docs/file%202.txt";*) //retourne "c:\\docs\\file 2.txt"
```

Convertir chemin systeme vers POSIX

Convertir chemin systeme vers POSIX (cheminSystème {; *}) -> Résultat

Paramètre	Type		Description
cheminSystème	Texte	➡	Chemin d'accès relatif ou absolu exprimé en syntaxe système
*	Opérateur	➡	Option d'encodage
Résultat	Texte	⮌	Chemin d'accès absolu exprimé en syntaxe POSIX

Description

La commande **Convertir chemin systeme vers POSIX** convertit un chemin d'accès exprimé avec la syntaxe système en chemin d'accès exprimé avec la syntaxe POSIX (Unix).

Passez dans le paramètre *cheminSystème* le chemin d'accès à un fichier ou un dossier, exprimé avec la syntaxe système (Mac OS ou Windows). Ce chemin peut être absolu ou relatif au dossier de la base (dossier contenant le fichier de structure de la base). Il n'est pas obligatoire que les éléments du chemin existent réellement sur le disque au moment de l'exécution de la commande (la commande ne teste pas la validité du chemin d'accès).

La commande retourne le chemin d'accès complet du fichier ou du dossier exprimé dans la syntaxe POSIX. La commande retourne toujours un chemin d'accès absolu, quel que soit le type de chemin passé dans *cheminSystème*. Si vous avez passé un chemin relatif dans *cheminSystème*, 4D complète la valeur retournée en ajoutant le chemin d'accès au dossier de la base.

Le paramètre optionnel * permet de définir l'encodage du chemin POSIX. Par défaut, **Convertir chemin systeme vers POSIX** n'encode pas les caractères spéciaux du chemin POSIX. Si vous passez le paramètre *, les caractères seront traduits (par exemple, "Mon dossier" devient "Mon%20dossier").

Exemple 1

Exemples sous OS X

```
$chemin:=Convertir chemin systeme vers POSIX("machd:file 2.txt")
    //machd est le disque de démarrage
    //retourne "/file 2.txt"
$chemin:=Convertir chemin systeme vers POSIX("disk2:file 2.txt")
    //disk2 est un disque additionnel (pas de démarrage)
    //retourne "/Volumes/disk2/file 2.txt"
$chemin:=Convertir chemin systeme vers POSIX("machd:file 2.txt";*)
    //retourne "/file%202.txt"
$chemin:=Convertir chemin systeme vers POSIX(":resources:images") //chemin relatif
    //retourne "/User/marc/Documents/basevideo/resources/images"
$chemin:=Convertir chemin systeme vers POSIX("resources:images") //chemin absolu
    //retourne "/resources/images"
```

Exemple 2

Exemple sous Windows

```
$chemin:=Convertir chemin systeme vers POSIX("c:\docs\file 2.txt")
    `retourne "c:/docs/file 2.txt"
$chemin:=Convertir chemin systeme vers POSIX("\\srv\tempo\file.txt")
    `retourne "//srv/tempo/file.txt"
```

COPIER DOCUMENT

COPIER DOCUMENT (nomSource ; nomDest {; nouvNom} {; *})

Paramètre	Type		Description
nomSource	Chaîne	⇒	Chemin d'accès du fichier ou du dossier à copier
nomDest	Chaîne	⇒	Nom ou chemin d'accès du fichier ou du dossier copié
nouvNom	Chaîne	⇒	Nouveau nom du fichier ou du dossier copié
*	Opérateur	⇒	Remplacer le document existant le cas échéant

Description

La commande **COPIER DOCUMENT** copie le fichier ou dossier désigné par *nomSource* à l'emplacement désigné par *nomDest* et le renomme optionnellement.

- **Copie de fichier**
  Dans ce cas, le paramètre *nomSource* peut contenir :
  - soit un chemin d'accès complet de fichier, exprimé par rapport à la racine du volume,
  - soit un chemin d'accès relatif au dossier de la base.

  Le paramètre *nomDest* peut contenir plusieurs types d'emplacements :
  - un chemin d'accès complet de fichier exprimé par rapport à la racine du volume : le fichier est recopié à cet emplacement
  - un nom de fichier ou un chemin d'accès de fichier relatif au dossier de la base : le fichier est recopié dans le dossier de la base (les sous-dossiers doivent exister)
  - un chemin d'accès de dossier complet ou relatif au dossier de la base (*nomDest* doit se terminer par un séparateur de dossier de la plate-forme) : le fichier est recopié dans le dossier désigné. Les dossiers doivent déjà exister sur le disque, il ne sont pas pas créés.

  Une erreur est générée si un document nommé *nomDest* existe déjà, à moins que vous n'ayez spécifié le paramètre optionnel *, qui indique à **COPIER DOCUMENT** de supprimer et de remplacer le document à l'emplacement de destination dans ce cas.

- **Copie de dossier**
  Pour indiquer que vous désignez un dossier, les chaînes passées dans *nomSource* et *nomDest* doivent se terminer par un séparateur de dossier de la plate-forme. Par exemple, sous Windows "C:\\Element\\" désigne un dossier et "C:\\Element" désigne un fichier.
  Pour recopier un dossier, passez son chemin d'accès complet dans *nomSource*. Ce dossier doit exister sur le disque.
  Lorsqu'un dossier est défini dans le paramètre *nomSource*, un dossier doit également être désigné dans le paramètre *nomDest*. Vous devez passer un chemin d'accès complet de dossier (dont chaque élément doit déjà exister sur le disque).
  Si un dossier du même nom que celui désigné par le paramètre *nomSource* existe déjà à l'emplacement défini par *nomDest* et n'est pas vide, 4D vérifie son contenu avant de copier les éléments. Une erreur est générée si un fichier du même nom existe déjà, à moins que vous n'ayez spécifié le paramètre optionnel *, qui indique à la commande de supprimer et de remplacer le document à l'emplacement de destination dans ce cas.

A noter que vous pouvez passer un fichier dans le paramètre *nomSource* et un dossier dans le paramètre *nomDest*, afin de copier un fichier dans un dossier.

Le paramètre optionnel *nouvNom*, s'il est passé, permet de renommer le document copié à son emplacement de destination (fichier ou dossier). Lorsqu'il est passé dans le contexte d'une copie de fichier, ce paramètre remplace le nom éventuellement passé via le paramètre *nomDest*.

Exemple 1

L'exemple suivant duplique un document dans son propre dossier :

```
COPIER DOCUMENT("C:\\DOSSIER\\LeDoc";"C:\\DOSSIER\\LeDoc2")
```

Exemple 2

L'exemple suivant copie un document dans le dossier de la base (dans la mesure où **C:\\DOSSIER** n'est pas le dossier de la base) :

```
COPIER DOCUMENT("C:\\DOSSIER\\LeDoc";"LeDoc")
```

Exemple 3

L'exemple suivant copie un document d'un volume vers un autre :

```
COPIER DOCUMENT("C:\\DOSSIER\\LeDoc";"F:\\Archives\\LeDoc.OLD")
```

Exemple 4

L'exemple suivant duplique un document dans son propre dossier, écrasant la précédente copie si elle existe :

```
COPIER DOCUMENT("C:\\DOSSIER\\LeDoc";"C:\\DOSSIER\\LeDoc2";*)
```

Exemple 5

Copie d'un fichier dans un dossier spécifique en conservant le même nom :

```
COPIER DOCUMENT("C:\\Projets\\NomDoc";"C:\\Projets\\")
```

Exemple 6

Copie d'un fichier dans un dossier spécifique en conservant le même nom et en remplaçant le document existant :

```
COPIER DOCUMENT("C:\\Projets\\NomDoc";"C:\\Projets\\"; *)
```

Exemple 7

Copie d'un dossier dans un autre dossier (les deux dossiers doivent exister sur le disque) :

```
COPIER DOCUMENT("C:\\Projets\\";"C\\Archives\\2011\\")
```

Exemple 8

Les exemples suivants créent différents fichiers et dossiers dans le dossier de la base (exemples Windows). Dans tous le cas, le dossier "dossier2" doit exister :

```
COPIER DOCUMENT("dossier1\\nom1";"dossier2\\")
    //crée le fichier "dossier2/nom1"

COPIER DOCUMENT("dossier1\\nom1";"dossier2\\" ; "nouveauté")
    //crée le fichier "dossier2/nouveauté"

COPIER DOCUMENT("dossier1\\nom1";"dossier2\\nom2")
    //crée le fichier "dossier2/nom2"

COPIER DOCUMENT("dossier1\\nom1";"dossier2\\nom2";"nouveauté")
    //crée le fichier "dossier2/nouveauté" (nom2 est ignoré)

COPIER DOCUMENT("dossier1\\" ; "dossier2\\")
    //crée le dossier "dossier2/dossier1/"

COPIER DOCUMENT("dossier1\\" ; "dossier2\\" ; "nouveauté")
    //crée le dossier "dossier2/nouveauté/"
```

Createur document

Createur document (nomFichier) -> Résultat

Paramètre	Type		Description
nomFichier	Chaîne	→	Nom de document ou Chemin d'accès complet à un document
Résultat	Chaîne	←	Chaîne vide (Windows) ou Créateur de fichier (Mac OS)

Description

La commande **Createur document** retourne le "créateur" du document dont vous avez passé le nom ou le chemin d'accès complet dans *document*.
Sous Windows, **Createur document** retourne une chaîne vide.

CREER ALIAS

CREER ALIAS (cheminCible ; cheminAlias)

Paramètre	Type		Description
cheminCible	Chaîne	⇒	Nom ou chemin d'accès de la cible de l'alias/du raccourci
cheminAlias	Chaîne	⇒	Nom ou chemin d'accès complet de l'alias/du raccourci à créer

Description

La commande **CREER ALIAS** crée un alias (appelé "raccourci" sous Windows) du fichier ou dossier cible désigné par le paramètre *cheminCible*, avec le nom et l'emplacement définis dans le paramètre *cheminAlias*.

Vous pouvez créer un alias de tout type de document ou de dossier. L'icône de l'alias sera identique à celle de l'élément cible. Elle comportera en outre une petite flèche et, sous Mac OS, le libellé de l'alias apparaîtra en caractères italiques.

La commande n'affecte pas de libellé par défaut à l'alias, vous devez passer un nom dans le paramètre *cheminAlias*. Si vous passez uniquement un nom dans ce paramètre, l'alias est créé dans le dossier actif courant (généralement, le dossier contenant le fichier de structure de la base).

Note : Sous Windows, les raccourcis sont des fichiers dont l'extension est ".LNK". Si vous ne passez pas cette extension, la commande l'ajoute automatiquement.

Si vous passez une chaîne vide dans *cheminCible*, la commande ne fait rien.

Exemple

Votre base génère des fichiers texte intitulés "RapportNuméro", stockés dans le dossier de la base. Vous souhaitez permettre à l'utilisateur de créer des raccourcis vers ces rapports et de les stocker où il le souhaite :

```
 `Méthode CREER_RAPPORT
C_TEXTE($vtRapport)
C_ALPHA(250;$vtChemin)
C_ALPHA(80;$vaNom)
C_HEURE(vDoc)
C_ENTIER($NumRapport)

$vTRapport:=... `Edition du rapport
$NumRapport:=... `Calcul du numéro du rapport
$vaNom:="Rapport"+Chaine($NumRapport)+".txt" `Nom du fichier
vDoc:=Creer document($vaNom)
Si(OK=1)
   ENVOYER PAQUET(vDoc;$vTRapport)
   FERMER DOCUMENT(vDoc)
 `Ajout de l'alias
   CONFIRMER("Créer un alias pour ce rapport ?)
   Si(OK=1)
      $vtChemin:=Selectionner dossier("Où souhaitez-vous créer l'alias ?")
      Si(OK=1)
         CREER ALIAS($vaNom;$vtChemin+$vaNom)
         Si(OK=1)
            MONTRER SUR DISQUE($vtChemin+$vaNom)
 `Visualisation de l'emplacement de l'alias
         Fin de si
      Fin de si
   Fin de si
Fin de si
```

Variables et ensembles système

La variable système OK prend la valeur 1 si la commande a été correctement exécutée, sinon elle prend la valeur 0.

Creer document

Creer document (nomFichier {; typeFichier}) -> Résultat

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Nom de document ou Chemin d'accès complet de document ou Chaîne vide pour afficher la boîte de dialogue standard d'enregistrement de fichiers
typeFichier	Chaîne	⇒	Liste des types de documents à filtrer, ou "*" pour ne pas filtrer les documents
Résultat	RefDoc	⇐	Numéro de référence du document

Description

La commande **Creer document** crée un document et retourne son numéro de référence de document.

Vous passez le nom ou le chemin d'accès complet du nouveau document dans *document*. Si *document* existe déjà, il est remplacé. Cependant, si le document est verrouillé ou est déjà ouvert, une erreur est générée.

Si vous passez une chaîne vide dans *document*, une boîte de dialogue standard d'enregistrement de fichiers apparaît et l'utilisateur peut spécifier le nom du document. Si dans ce cas l'utilisateur clique sur le bouton **Annuler**, **Creer document** retourne une référence de document nulle, et la variable OK prend la valeur 0.

Creer document crée par défaut un document de type TEXT (Mac OS) ou .TXT (Windows). Pour créer un autre type de document, passez un type dans le paramètre optionnel *typeFichier*.

Si vous utilisez la boîte de dialogue standard d'enregistrement de fichiers, vous pouvez passer dans le paramètre *typeFichier* un ou plusieurs type(s) de fichier(s) afin de configurer la liste des types autorisés dans la boîte de dialogue. Vous pouvez passer une liste de plusieurs types séparés par un ; (point virgule). Pour chaque type défini, une ligne sera ajoutée dans le menu de choix de type de la boîte de dialogue.
Sous Mac OS, vous pouvez passer soit type Mac OS classique (TEXT, APPL, etc.), soit un type UTI (Uniform Type Identifier). Les types UTIs ont été définis par Apple afin de répondre aux besoins d'uniformisation des types de fichiers. Par exemple, "public.text" est le type UTI des fichiers de type texte. Pour plus d'informations sur les UTIs, reportez-vous à l'adresse https://developer.apple.com/library/mac/#documentation/Miscellaneous/Reference/UTIRef/Articles/System-DeclaredUniformTypeIdentifiers.html (documentation en anglais).
Sous Windows, vous pouvez également passer un type de fichier classique Mac OS — 4D effectue la correspondance en interne — ou l'extension des fichiers (.txt, .exe, etc.). A noter que sous Windows, l'utilisateur aura la possibilité de "forcer" l'affichage de tous les types de fichiers en saisissant *.* dans la boîte de dialogue. Toutefois dans ce cas, 4D effectuera une vérification supplémentaire des types des fichiers sélectionnés : si l'utilisateur sélectionne un type de fichier non autorisé, la commande retourne une erreur.
Si vous ne souhaitez pas restreindre les fichiers affichés à un ou plusieurs types, passez la chaîne "*" (étoile) ou ".*" dans *typeFichier*.
Sous Windows, vous pouvez passer une extension de fichier Windows ou un type de fichier Mac OS associé à l'aide de la commande **ASSOCIER TYPES FICHIER**. Si vous souhaitez créer un document sans extension, un document comportant plusieurs extensions, ou un document comportant une extension de plus de trois caractères, n'utilisez pas le paramètre *typeFichier* et passez le nom complet dans *document* (cf. exemple 2).

Si le document est correctement créé et ouvert, **Creer document** retourne sa référence de document et la variable système OK prend la valeur 1. La variable système Document est mise à jour et retourne le chemin d'accès complet du document créé.

Une fois que vous avez créé et ouvert un document, vous pouvez écrire ou lire des valeurs dans ce document à l'aide des commandes **RECEVOIR PAQUET** et **ENVOYER PAQUET**, que vous pouvez combiner avec les commandes **Position dans document** et **CHANGER POSITION DANS DOCUMENT** pour accéder directement à certains endroits du document.

N'oubliez pas d'appeler finalement **FERMER DOCUMENT** pour le document.

Exemple 1

L'exemple suivant crée et ouvre un nouveau document qui s'appelle "Note", écrit la chaîne "Bonjour" et le referme :

```
C_HEURE(vDoc)
vDoc:=Creer document("Note.txt") ` Créer un nouveau document qui s'appelle Note
Si(OK=1)
   ENVOYER PAQUET(vDoc;"Bonjour") ` Ecrire un mot dans le document
   FERMER DOCUMENT(vDoc) ` Fermer le document
Fin de si
```

Exemple 2

L'exemple suivant crée sous Windows des documents avec des extensions non standard :

```
$vhMonDoc:=Creer document("LeDoc.ext1.ext2") `Plusieurs extensions
$vhMonDoc:=Creer document("LeDoc.shtml") `Extension longue
$vhMonDoc:=Creer document("LeDoc.") `Pas d'extension (le point "." est obligatoire)
```

Variables et ensembles système

Si le document est correctement créé, la variable système OK prend la valeur 1 et la variable système Document contient le chemin d'accès et le nom du fichier *document*.

CREER DOSSIER

CREER DOSSIER (cheminAccès {; *})

Paramètre	Type		Description
cheminAccès	Chaîne	⇒	Chemin d'accès au nouveau dossier à créer
*	Opérateur	⇒	Créer la hiérarchie du dossier

Description

La commande **CREER DOSSIER** crée un dossier en fonction du chemin d'accès que vous passez dans le paramètre *cheminAccès*.

Si vous passez un nom dans *cheminAccès*, le dossier est créé dans le dossier de la base.

Vous pouvez également passer dans *cheminAccès* une hiérarchie de dossiers à partir de la racine du volume ou du dossier de la base (dans ce cas, la chaîne doit se terminer par un séparateur de dossier).
Si vous omettez le paramètre *, une erreur est générée et aucun dossier n'est créé si au moins un dossier intermédiaire n'existe pas.
Si vous passez le paramètre *, **CREER DOSSIER** recrée la hiérarchie de dossiers si nécessaire et aucune erreur n'est générée. Dans ce cas, vous pouvez également passer un chemin d'accès de document dans *cheminAccès*. Le nom du document est alors ignoré mais la hiérarchie de dossiers définie dans *cheminAccès* est créée récursivement.

Exemple 1

L'exemple suivant crée le dossier “Archives” dans le dossier de la base :

```
CREER DOSSIER("Archives")
```

Exemple 2

L'exemple suivant crée le dossier “Archives” dans le dossier de la base, puis crée les sous-dossiers “Janvier” et “Février”:

```
CREER DOSSIER("Archives")
CREER DOSSIER("Archives\\Janvier")
CREER DOSSIER("Archives\\Février")
```

Exemple 3

L'exemple suivant crée le dossier “Archives” à la racine du volume C :

```
CREER DOSSIER("C:\\Archives")
```

Exemple 4

Création de la hiérarchie de dossiers "C:\Archives\2011\January\" :

```
CREER DOSSIER("C:\\Archives\\2011\\January\\";*)
```

Exemple 5

Création du sous-dossier "\February\" dans le dossier existant "C:\Archives\" :

```
CREER DOSSIER("C:\\Archives\\2011\\February\\Doc.txt";*)
 // le fichier "Doc.txt" est ignoré
```

DEPLACER DOCUMENT

DEPLACER DOCUMENT (cheminSource ; cheminDest)

Paramètre	Type		Description
cheminSource	Chaîne	➡	Chemin d'accès complet au document existant
cheminDest	Chaîne	➡	Chemin d'accès de destination

Description

La commande **DEPLACER DOCUMENT** déplace ou renomme un document.

Vous passez le chemin d'accès complet au document existant dans le paramètre *cheminSource* et le nouveau nom et/ou emplacement du document dans *cheminDest*.

Attention : Avec **DEPLACER DOCUMENT**, vous pouvez déplacer un document depuis et vers tous les dossiers du même volume. Si vous souhaitez déplacer un document entre deux volumes différents, utilisez la commande **COPIER DOCUMENT** pour "déplacer" le document puis effacez le document original avec la commande **SUPPRIMER DOCUMENT**.

Exemple 1

L'exemple suivant renomme le document **DocNom** :

```
DEPLACER DOCUMENT("C:\\DOSSIER\\DocNom";"C:\\DOSSIER\\NouveauDocNom")
```

Exemple 2

L'exemple suivant déplace et renomme le document **DocNom** :

```
DEPLACER DOCUMENT("C:\\DOSSIER1\\DocNom";"C:\\DOSSIER2\\NouveauDocNom")
```

Exemple 3

L'exemple suivant déplace le document **DocNom** :

```
DEPLACER DOCUMENT("C:\\DOSSIER1\\DocNom";"C:\\DOSSIER2\\DocNom")
```

Note : Dans les deux derniers exemples, le dossier de destination *"C:\\DOSSIER2"* doit déjà exister. En effet, la commande **DEPLACER DOCUMENT** déplace uniquement un document, elle ne peut créer de dossiers.

Document vers texte

Document vers texte (nomFichier {; jeuCaractères {; modeRetour}}) -> Résultat

Paramètre	Type		Description
nomFichier	Chaîne	➡	Nom de document ou Chemin d'accès à un document
jeuCaractères	Texte, Entier long	➡	Nom ou Numéro de jeu de caractères
modeRetour	Entier long	➡	Mode de traitement des retours à la ligne
Résultat	Texte	⬅	Texte issu du document

Description

La commande **Document vers texte** permet de récupérer directement le contenu d'un fichier sur disque dans une variable texte ou un champ texte 4D.

Passez dans *nomFichier* le nom ou le chemin d'accès du fichier à lire. Le fichier doit exister sur le disque, sinon une erreur est générée. Vous pouvez passer :

- uniquement le nom du fichier, par exemple "monFichier.txt" : dans ce cas, le fichier doit se trouver à côté du fichier de structure de l'application.
- un chemin d'accès relatif au fichier de structure de l'application, par exemple "\\docs\\monFichier.txt" sous Windows ou ":docs:monFichier.txt" sous OS X.
- un chemin d'accès absolu, par exemple "c:\\app\\docs\\monFichier.txt" sous Windows ou "MacHD:docs:monFichier.txt" sous OS X.

Vous pouvez passer dans *jeuCaractères* le jeu de caractères à utiliser pour la lecture. Vous pouvez passer une chaîne contenant le nom standard du jeu (par exemple "ISO-8859-1" ou "UTF-8") ou son identifiant MIBEnum (entier long). Pour plus d'informations sur la liste des jeux de caractères pris en charge par 4D, reportez-vous à la description de la commande **CONVERTIR DEPUIS TEXTE**.

Si le document contient une BOM (Byte Order Mark), 4D utilise le jeu de caractères qu'elle définit au lieu du jeu de caractères spécifié dans *jeuCaractères* (ce paramètre est alors ignoré).

Si le document ne contient pas de BOM et si le paramètre *jeuCaractères* est omis, 4D utilise par défaut les jeux de caractères suivants :

- sous Windows : ANSI
- sous OS X : MacRoman

Vous pouvez passer dans *modeRetour* un entier long indiquant le traitement à effectuer sur les caractères de fin de ligne présents dans le document. Vous pouvez utiliser une des constantes suivantes, placées dans le thème "**Documents système**" :

Constante	Type	Valeur	Comment
Document avec CR	Entier long	3	Les fins de ligne sont convertis au format OS X : CR (*carriage return*)
Document avec CRLF	Entier long	2	Les fins de ligne sont convertis au format Windows : CRLF (*carriage return* + *line feed*)
Document avec format natif	Entier long	1	(Défaut) Les fins de ligne sont convertis au format natif de la plate-forme d'exécution : CR (*carriage return*) sous OS X, CRLF (*carriage return* + *line feed*) sous Windows
Document avec LF	Entier long	4	Les fins de ligne sont convertis au format Unix : LF (*line feed*)
Document inchangé	Entier long	0	Aucun traitement

Par défaut, si le paramètre *modeRetour* est omis, les caractères de fin de ligne sont traités en mode natif (1).

Note : Cette commande ne modifie pas la variable OK. En cas d'échec, une erreur est générée, que vous pouvez intercepter à l'aide d'une méthode installées par la commande **APPELER SUR ERREUR**.

Exemple

Soit le document texte suivant (les champs sont séparés par des tabulations) :

```
id    name    price    vat
3    4D Tags    99    19,6
```

Si vous exécutez ce code :

```
$Text:=Document vers texte("products.txt")
```

... vous obtenez :

```
// $Text = "id\tname\tprice\tvat\r\n3\t4D Tags\t99 \t19,6"
// \t = tabulation
// \r = CR
```

FERMER DOCUMENT

FERMER DOCUMENT (docRef)

Paramètre	Type		Description
docRef	RefDoc	⇨	Numéro de référence du document

Description

FERMER DOCUMENT ferme le document spécifié par *docRef*.

Fermer un document est le seul moyen de s'assurer que les données écrites dans le fichier sont sauvegardées. Vous devez fermer tous les documents ouverts par les commandes **Ouvrir document**, **Creer document** et **Ajouter a document**.

Exemple

L'exemple suivant permet à l'utilisateur de créer un nouveau document, écrit la chaîne “Bonjour”, puis le referme :

```
C_HEURE(vDoc)
vDoc:=Creer document("")
Si(OK=1)
   ENVOYER PAQUET(vDoc;"Bonjour") ` Ecrire un mot dans le document
   FERMER DOCUMENT(vDoc) ` Fermer le document
Fin de si
```

Lire chemin document localise

Lire chemin document localise (cheminRelatif) -> Résultat

Paramètre	Type		Description
cheminRelatif	Texte	➡	Chemin d'accès relatif du document dont on veut obtenir la version localisée
Résultat	Texte	➲	Chemin d'accès absolu du document localisé

Description

La commande **Lire chemin document localise** retourne le chemin d'accès complet (absolu) d'un document désigné par *cheminRelatif* et situé dans un dossier xxx.lproj.

Cette commande doit être utilisée dans le cadre d'une architecture d'application multi-langue basée sur la présence d'un dossier **Resources** et de sous-dossiers *xxx.lproj* (xxx représentant une langue). Avec cette architecture, 4D prend automatiquement en charge les fichiers localisés de type .xliff ainsi que les images, mais vous pouvez avoir besoin d'utiliser le même mécanisme pour d'autres types de fichiers.

Passez dans *cheminRelatif* le chemin d'accès relatif du document recherché. Le chemin saisi doit être relatif au premier niveau d'un dossier "xxx.lproj" de la base. La commande retournera un chemin d'accès complet en utilisant le dossier "xxx.lproj" correspondant à la langue courante de la base.

Note : La langue courante est définie soit automatiquement par 4D en fonction du contenu du dossier **Resources** (cf. commande **Lire langue base**), soit via la commande **FIXER LANGUE BASE**.

Vous pouvez exprimer le contenu du paramètre *cheminRelatif* à l'aide d'une syntaxe posix ou système. Par exemple :

- xsl/log.xsl (syntaxe posix : utilisable sous Mac OS ou Windows)
- xsl\log.xsl (Windows uniquement)
- xsl:log.xsl (Mac OS uniquement)

Le chemin d'accès absolu retourné par la commande est toujours exprimé en syntaxe système.

4D Server : En mode distant, la commande retourne le chemin du dossier **Resources** sur le poste client si la commande est appelée depuis un process client.

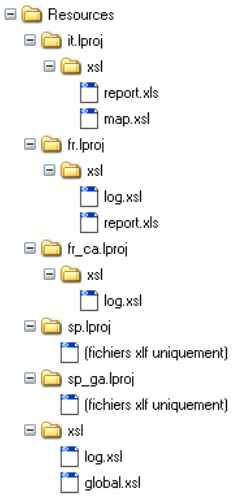
4D recherche le fichier en respectant une séquence permettant de traiter tous les cas d'applications multi-langues. A chaque étape, 4D teste la présence de *cheminRelatif* dans le dossier correspondant à la langue et retourne le chemin complet en cas de succès. Si *cheminRelatif* n'est pas trouvé ou si le dossier n'existe pas, 4D passe à l'étape suivante. Voici les dossiers des étapes de recherche :

Langue courante (ex : fr-ca)
Langue courante sans la région (ex : fr)
Langue chargée par défaut au démarrage (ex : es-ga)
Langue chargée par défaut au démarrage sans la région (ex : es)
Premier dossier .lproj trouvé (ex : it.lproj)
Premier niveau du dossier Resources

Si *cheminRelatif* n'est trouvé à aucun de ces emplacements, la commande retourne une chaîne vide.

Exemple

Dans le but de transformer un fichier xml en html, vous souhaitez utiliser un fichier de transformation "log.xsl". Ce fichier diffère suivant la langue courante. Vous souhaitez donc connaître le chemin du fichier "log.xsl" à utiliser.
Voici le contenu du dossier Resources :



Pour utiliser un fichier .xsl adapté à la langue courante, il vous suffit de passer :

```
$monxsl:=Lire chemin document localise("xsl/log.xsl")
```

Si la langue courante est, par exemple, le français canadien (fr-ca), la commande retourne :

- sous Windows : C:\users\…\…\…\resources\fr_ca.lproj\xsl\log.xsl"
- sous Mac OS : "HardDisk:users:…:…:…:resources:fr_ca.lproj:xsl:log.xsl"

LIRE ICONE DOCUMENT

LIRE ICONE DOCUMENT (cheminDoc ; icône {; taille})

Paramètre	Type		Description
cheminDoc	Chaîne	⇨	Nom ou chemin d'accès du fichier duquel obtenir l'icône ou chaîne vide pour afficher la boîte de dialogue d'ouverture de fichiers
icône	Image	⇨	Champ ou variable image
taille	Entier long	⇨	Taille de l'icône (en pixels)

Description

La commande **LIRE ICONE DOCUMENT** retourne dans le champ ou la variable image 4D *icône*, l'icône du document dont vous avez passé le nom ou le chemin d'accès complet dans *cheminDoc*. *cheminDoc* peut désigner un fichier de tout type (document, exécutable, raccourci ou alias...) ou un dossier.

Passez dans *cheminDoc* le chemin d'accès absolu du document dont vous souhaitez récupérer l'icône. Vous pouvez passer uniquement le nom du document ou un chemin d'accès relatif, dans ce cas il doit se trouver dans le dossier courant de la base (généralement, le dossier contenant le fichier de structure de la base).
Si vous passez une chaîne vide dans *cheminDoc*, la boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de désigner un fichier. Une fois la boîte de dialogue validée, la variable système Document contient le chemin d'accès complet du fichier sélectionné.

Passez dans le paramètre *icône* un champ ou une variable image 4D. Après l'exécution de la commande, ce paramètre contient l'icône du fichier (au format PICT).

Le paramètre optionnel *taille* vous permet d'indiquer les dimensions de l'image que vous souhaitez obtenir. La valeur du paramètre correspond à la longueur d'un côté du carré dans lequel l'image sera incluse. Généralement, les icônes sont définies en 32x32 pixels ("grande icône") ou 16x16 pixels ("petite icône"). Si vous passez 0 ou omettez le paramètre, la commande retourne l'icône dans sa plus grande taille disponible.

LISTE DES DOCUMENTS

LISTE DES DOCUMENTS (cheminAccès ; documents {; options})

Paramètre	Type		Description
cheminAccès	Chaîne	⇨	Chemin d'accès de volume ou de dossier
documents	Tableau texte	⇦	Nom des documents situés à cet endroit
options	Entier long	⇨	Options de construction de la liste

Description

La commande **LISTE DES DOCUMENTS** remplit le tableau de type Texte *documents* avec les noms des documents situés à l'endroit que vous avez indiqué avec le paramètre *cheminAccès*.

Note : Vous devez passer un chemin d'accès absolu dans le paramètre *cheminAccès*.

Par défaut, si vous omettez le paramètre *options*, seuls les noms des documents sont retournés dans le tableau *documents*. Vous pouvez modifier ce fonctionnement en passant dans le paramètre *options* une ou plusieurs des constantes suivantes, placées dans le thème **Documents système** :

Constante	Type	Valeur	Comment
Chemin absolu	Entier long	2	Le tableau *documents* contient des chemins d'accès absolus
Chemin POSIX	Entier long	4	Le tableau *documents* contient des chemins d'accès au format POSIX
Chemin récursif	Entier long	1	Le tableau *documents* contient les fichiers et tous les sous-dossiers du dossier spécifié
Ignorer invisibles	Entier long	8	Les documents invisibles ne sont pas listés

Notes :

- Avec l'option Chemin récursif en mode relatif (option 1 seule), les chemins des documents situés dans des sous-dossiers débutent par les caractères ":" ou "\" en fonction de la plate-forme.
- Avec l'option Chemin POSIX en mode relatif (option 4 seule), les chemins ne débutent pas par "/"
- Avec l'option Chemin POSIX en mode absolu (option 4 + 2), les chemins débutent toujours par "/"

S'il n'y pas de document à l'endroit défini, la commande retourne un tableau vide. Si le chemin d'accès que vous avez passé dans *cheminAccès* est invalide, **LISTE DES DOCUMENTS** génère une erreur de gestionnaire de fichier que vous pouvez intercepter à l'aide d'une méthode installée par **APPELER SUR ERREUR**.

Exemple 1

Liste de tous les documents dans un dossier (syntaxe par défaut) :

```
LISTE DES DOCUMENTS("C:\\";tabFichiers)
```

-> tabFichiers :
Texte1.txt
Texte2.txt

Exemple 2

Liste de tous les documents dans un dossier en mode absolu :

```
LISTE DES DOCUMENTS("C:\\";tabFichiers; Chemin absolu)
```

-> tabFichiers :
C:\Texte1.txt
C:\Texte2.txt

Exemple 3

Liste de tous les documents en mode récursif (relatif) :

```
LISTE DES DOCUMENTS("C:\\";tabFichiers;Chemin récursif)
```

-> tabFichiers :
Texte1.txt
Texte2.txt
\Dossier1\Texte3.txt
\Dossier1\Texte4.txt
\Dossier2\Texte5.txt
\Dossier2\Dossier3\Image1.png

Exemple 4

Liste de tous les documents en mode récursif absolu :

```
LISTE DES DOCUMENTS("C:\\MonDossier\\";tabFichiers;Chemin récursif+Chemin absolu)
```

-> tabFichiers :
C:\MonDossier\MonTexte1.txt
C:\MonDossier\MonTexte2.txt
C:\MonDossier\Dossier1\MonTexte3.txt
C:\MonDossier\Dossier1\MonTexte4.txt
C:\MonDossier\Dossier2\MonTexte5.txt
C:\MonDossier\Dossier2\Dossier3\MonImage1.png

Exemple 5

Liste de tous les documents en mode récursif POSIX (relatif) :

```
LISTE DES DOCUMENTS("C:\\MonDossier\\";tabFichiers;Chemin récursif+Chemin POSIX)
```

-> tabFichiers :
MonTexte1.txt
MonTexte2.txt
Dossier1/MonTexte3.txt
Dossier1/MonTexte4.txt
Dossier2/MonTexte5.txt
Dossier2/Dossier3/MonImage1.png

LISTE DES DOSSIERS

LISTE DES DOSSIERS (cheminAccès ; dossiers)

Paramètre	Type		Description
cheminAccès	Chaîne	⇨	Chemin d'accès de volume, répertoire ou dossier
dossiers	Tableau chaîne	⇦	Noms des dossiers situés à cet endroit

Description

La commande **LISTE DES DOSSIERS** remplit le tableau de type Texte ou Alpha *dossiers* avec les noms des dossiers (répertoires sous Windows) situés à l'endroit que vous avez indiqué avec le paramètre *cheminAccès*.

Note : Vous devez passer un chemin d'accès absolu dans le paramètre *cheminAccès*.

S'il n'y pas de dossier à cet endroit, la commande retourne un tableau vide. Si le chemin d'accès que vous avez passé dans *cheminAccès* est invalide, **LISTE DES DOSSIERS** génère une erreur de gestionnaire de fichiers que vous pouvez intercepter à l'aide d'une méthode installée par **APPELER SUR ERREUR**.

LISTE DES VOLUMES

LISTE DES VOLUMES (volumes)

Paramètre	Type		Description
volumes	Tableau chaîne	⇦	Noms des volumes actuellement montés

Description

LISTE DES VOLUMES remplit le tableau *volumes*, de type texte, avec les noms des volumes définis (Windows) ou montés (Mac OS) sur votre machine.

- Sous Mac OS, elle retourne la liste des volumes visibles au niveau du Finder. Seuls les noms des volumes sont retournés (par exemple "MacHD", "BootCamp"...).
- Sous Windows, elle retourne la liste des volumes couramment définis, même si le volume n'est pas physiquement présent (par exemple un volume "**E:**" sera retourné même s'il n'y a pas de CD ou de DVD dans le lecteur). Les noms des volumes sont suivis du caractère séparateur de dossiers ("C:\").

Exemple

A l'aide de la zone de défilement *ttVolumes*, vous voulez afficher la liste des volumes définis ou montés sur votre machine :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TABLEAU TEXTE(ttVolumes;0)
      LISTE DES VOLUMES(ttVolumes)

 // ...
Fin de cas
```

MONTRER SUR DISQUE

MONTRER SUR DISQUE (cheminAccès {; *})

Paramètre	Type		Description
cheminAccès	Chaîne	➡	Chemin d'accès de l'élément à montrer
*		➡	Si l'élément est un dossier, montrer son contenu

Description

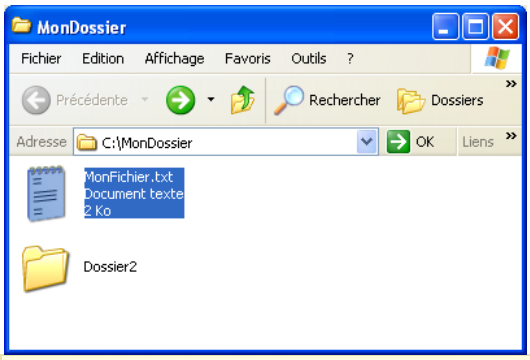
La commande **MONTRER SUR DISQUE** affiche dans une fenêtre standard du système d'exploitation le fichier ou le dossier dont le chemin d'accès est passé dans le paramètre *cheminAccès*.
Dans le cadre d'une interface utilisateur, cette commande permet à l'utilisateur de visualiser l'emplacement d'un fichier ou d'un dossier spécifique.

Par défaut, si *cheminAccès* désigne un dossier, la commande affiche le niveau du dossier lui-même. Si vous passez le paramètre facultatif *, la commande ouvre le dossier et affiche son contenu dans la fenêtre. Si *cheminAccès* désigne un fichier, le paramètre * est ignoré.

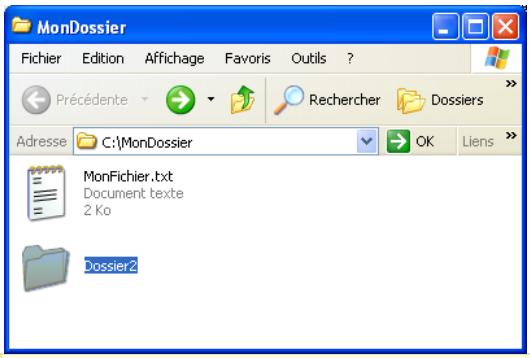
Exemple

Ces exemples illustrent le fonctionnement de la commande.

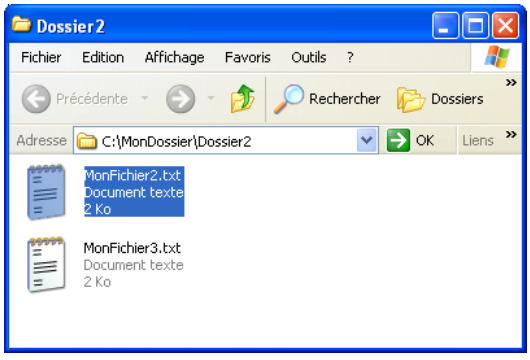
```
MONTRER SUR DISQUE("c:\\MonDossier\\MonFichier.txt") `Affiche le fichier désigné
```




```
MONTRER SUR DISQUE("c:\\MonDossier\\Dossier2") `Affiche le dossier désigné
```




```
MONTRER SUR DISQUE("c:\\MonDossier\\Dossier2";*) `Affiche le contenu du dossier désigné
```



Variables et ensembles système

La variable système OK prend la valeur 1 si la commande est correctement exécutée, sinon elle prend la valeur 0.

Ouvrir document

Ouvrir document (nomFichier {; typeFichier}{; mode}) -> Résultat

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Nom du document ou Chemin d'accès complet au document ou Chaîne vide pour afficher la boîte de dialogue
typeFichier	Chaîne	⇒	Liste des types de documents à filtrer, ou "*" pour ne pas filtrer les documents
mode	Entier long	⇒	Mode d'ouverture du document
Résultat	RefDoc	⇐	Numéro de référence du document

Description

La commande **Ouvrir document** ouvre le document dont vous avez passé le nom dans *nomFichier*.

Si vous passez une chaîne vide ("") dans *nomFichier*, une boîte de dialogue standard d'ouverture de fichiers apparaît et l'utilisateur peut désigner le document. Si dans ce cas l'utilisateur clique sur le bouton **Annuler**, aucun document n'est ouvert, **Ouvrir document** retourne une référence de document nulle, et la variable OK prend la valeur 0.

- Si le document est correctement ouvert, **Ouvrir document** retourne sa référence de document et la variable OK prend la valeur 1.
- Si le document était déjà ouvert en lecture et que le paramètre *mode* est omis, **Ouvrir document** l'ouvre en mode lecture/écriture par défaut et la variable OK prend la valeur 1.
- Si le document était déjà ouvert en écriture et que vous tentez de l'ouvrir en mode écriture, une erreur -43 est générée. En revanche, vous pouvez l'ouvrir en mode lecture dans ce cas, la variable OK prend la valeur 1.
- Si le document n'existe pas, une erreur est générée.

Passez dans le paramètre *typeFichier* le ou les type(s) de fichier(s) pouvant être sélectionnés dans la boîte de dialogue d'ouverture. Vous pouvez passer une liste de plusieurs types séparés par un ; (point virgule). Pour chaque type défini, une ligne sera ajoutée dans le menu de choix de type de la boîte de dialogue.

- Sous Mac OS, vous pouvez passer soit un type Mac OS classique (TEXT, APPL, etc.), soit un type UTI (Uniform Type Identifier). Les types UTIs ont été définis par Apple afin de répondre aux besoins d'uniformisation des types de fichiers. Par exemple, "public.text" est le type UTI des fichiers de type texte. Pour plus d'informations sur les UTIs, reportez-vous à l'adresse : https://developer.apple.com/library/mac/#documentation/FileManagement/Conceptual/understanding_utis/understand_utis_intro/understand_utis_intro.html (documentation en anglais).

- Sous Windows, vous pouvez également passer un type de fichier classique Mac OS — 4D effectue la correspondance en interne — ou l'extension des fichiers (.txt, .exe, etc.). A noter que sous Windows, l'utilisateur aura la possibilité de "forcer" l'affichage de tous les types de fichiers en saisissant *.* dans la boîte de dialogue. Toutefois dans ce cas, 4D effectuera une vérification supplémentaire des types des fichiers sélectionnés : si l'utilisateur sélectionne un type de fichier non autorisé, la commande retourne une erreur.

Si vous ne souhaitez pas restreindre les fichiers affichés à un ou plusieurs types, passez la chaîne "*" (étoile) ou ".*" dans *typeFichier*.

Le paramètre optionnel *mode* permet de définir le mode d'ouverture du fichier document. Quatre modes d'ouverture sont disponibles. 4D vous propose les constantes prédéfinies suivantes, placées dans le thème **Documents système** :

Constante	Type	Valeur
Lecture et écriture	Entier long	0
Lire chemin accès	Entier long	3
Mode écriture	Entier long	1
Mode lecture	Entier long	2

Lorsqu'un document est ouvert, **Ouvrir document** se place initialement au début du document, alors que **Ajouter a document** se place à la fin.

Une fois que vous avez ouvert un document, vous pouvez écrire ou lire des valeurs dans ce document à l'aide des commandes **RECEVOIR PAQUET** et **ENVOYER PAQUET**, que vous pouvez combiner avec les commandes **Position dans document** et **CHANGER POSITION DANS DOCUMENT** pour accéder directement à certains endroits du document.

N'oubliez pas d'appeler finalement **FERMER DOCUMENT** pour le document.

Exemple 1

L'exemple suivant ouvre un document existant qui s'appelle "Note", écrit la chaîne "Au revoir" dans le document et le referme. Si le document contient déjà la chaîne "Bonjour", elle est remplacée :

```
C_HEURE(vDoc)
vDoc:=Ouvrir document("Note.txt";Lecture et écriture) ` Ouvrir le document Note
Si(OK=1)
   ENVOYER PAQUET(vDoc;"Au revoir") ` Ecrire un mot dans le document
   FERMER DOCUMENT(vDoc) ` Fermer le document
Fin de si
```

Exemple 2

Vous pouvez lire un document déjà ouvert en écriture :

```
vDoc:=Ouvrir document("PassFile";"TEXT") ` Le fichier est ouvert
vRef:=Ouvrir document("PassFile";"TEXT";Mode lecture) ` Le fichier est lu
```

Variables et ensembles système

Si le document est correctement ouvert, la variable système OK prend la valeur 1, sinon elle prend la valeur 0. Après l'appel, la variable système Document contient le nom complet du document.
Si vous passez la valeur 3 dans *mode*, la fonction retourne ?00:00:00? (pas de référence de document). Le document n'est pas ouvert mais les variables système Document et OK sont mises à jour :

- OK prend la valeur 1,
- Document contient le chemin d'accès et le nom du fichier *document*.

Note : Si vous passez une chaîne vide dans *document*, une boîte de dialogue d'ouverture de fichiers apparaît. Si elle est validée, Document et OK sont mises à jour comme décrit ci-dessus. Si elle est annulée, OK prend la valeur 0.

Position dans document

Position dans document (docRef) -> Résultat

Paramètre	Type		Description
docRef	RefDoc	→	Numéro de référence de document
Résultat	Réel	←	Position dans le fichier (exprimée en octets) à partir du début du fichier

Description

Cette commande ne fonctionne qu'avec un document déjà ouvert, dont vous avez passé le numéro de référence dans le paramètre *docRef*.

Position dans document retourne la position, à partir du début du document, à laquelle la prochaine lecture (**RECEVOIR PAQUET**) ou écriture (**ENVOYER PAQUET**) aura lieu.

PROPRIETES DOCUMENT

PROPRIETES DOCUMENT (nomFichier ; verrouillé ; invisible ; créé le ; créé à ; modifié le ; modifié à)

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Nom du document
verrouillé	Booléen	⇐	Verrouillé (Vrai) ou non verrouillé (Faux)
invisible	Booléen	⇐	Invisible (Vrai) ou visible (Faux)
créé le	Date	⇐	Date de création
créé à	Heure	⇐	Heure de création
modifié le	Date	⇐	Date de la dernière modification
modifié à	Heure	⇐	Heure de la dernière modification

Description

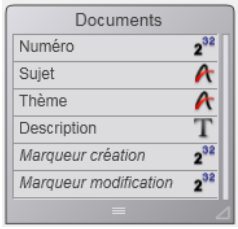
La commande **PROPRIETES DOCUMENT** retourne des informations sur le document dont le nom ou le chemin d'accès est passé dans le paramètre *document*.

Après l'appel :

- *verrouillé* retourne Vrai si le document est verrouillé. Un document verrouillé ne peut pas être modifié.
- *invisible* retourne Vrai si le document est caché.
- *créé le* et *créé à* retournent la date et l'heure de création du document.
- *modifié le* et *modifié à* retournent la date et l'heure de la dernière modification du document.

Exemple

Vous avez créé une base de documentation et vous voulez exporter tous les enregistrements créés dans le base vers un document sur disque. Comme la base est régulièrement mise à jour, vous voulez écrire un algorithme d'export qui crée ou recrée chaque document sur disque si le document n'existe pas ou si l'enregistrement correspondant a été modifié depuis la dernière sauvegarde du document. Par conséquent, vous devez comparer la date et l'heure de modification du document (s'il existe) avec celles de l'enregistrement correspondant. Pour illustrer cet exemple, nous allons utiliser la table suivante :



Plutôt que de sauvegarder une date et une heure dans chaque enregistrement, vous pouvez stocker un "marqueur" dont la valeur exprime le nombre de secondes écoulées depuis une date antérieure arbitraire (dans cet exemple, le 1er janvier 1995 à 00:00:00) ainsi que la date et l'heure de la sauvegarde de l'enregistrement.

Dans notre exemple, le champ *[Documents]Marqueur création* contient le marqueur de création de l'enregistrement et le champ *[Documents]Marqueur modification* contient le marqueur de la dernière modification de l'enregistrement.

La méthode projet ***marqueurTemps*** suivante calcule le marqueur de temps par rapport à une date et une heure spécifiques ou par rapport à la date et l'heure courantes si aucun paramètre n'est passé :

```
  ` Méthode projet marqueurTemps
  ` marqueurTemps { ( Date ; Heure ) } -> Entier long
  ` marqueurTemps { ( Date ; Heure ) } -> Nombre de secondes depuis le 1er janvier 1995

C_DATE($1;$vdDate)
C_HEURE($2;$vhTime)
C_ENTIER LONG($0)

Si(Nombre de paramètres=0)
   $vdDate:=Date du jour
   $vhTime:=Heure courante
Sinon
   $vdDate:=$1
   $vhTime:=$2
Fin de si
$0:=(($vdDate-!01/01/95!)*86400)+$vhTime
```

Note : Avec cette méthode, vous pouvez encoder toutes les dates et les heures situées entre le *01/01/95* à *00:00:00* et le *19/01/2063* à *03:14:07*, ce qui représente l'intervalle de données exploitables par un entier long (de *0* à *2^31* moins 1).

A l'inverse, les méthodes projet ***Marqueur vers date*** et ***Marqueur vers heure*** vous permettent d'extraire la date et l'heure stockées dans un marqueur :

```
  ` Méthode projet Marqueur vers date
  ` Marqueur vers date ( Entier long ) -> Date
  ` Marqueur vers date ( Marqueur ) -> Date extraite

C_DATE($0)
C_ENTIER LONG($1)

$0:=!01/01/95!+($1\86400)
```

```
  ` Méthode projet Marqueur vers heure
  ` Marqueur vers heure ( Entier long ) -> Heure
  ` Marqueur vers heure ( Marqueur ) -> Heure extraite

C_HEURE($0)
C_ENTIER LONG($1)

$0:=Heure(Chaine heure(†00:00:00†+($1%86400)))
```

Pour vous assurer que les marqueurs des enregistrements sont correctement mis à jour, quelle que soit la manière dont ils sont créés ou modifiés, il suffit de faire appliquer cette règle par le trigger de la table *[Documents]*:

```
 // Trigger de la table [Documents]

Au cas ou
   :(Evenement trigger=Sur sauvegarde nouvel enreg)
      [Documents]Marqueur création:=marqueurTemps
      [Documents]Marqueur modification:=marqueurTemps
   :(Evenement trigger=Sur sauvegarde enregistrement)
      [Documents]Marqueur modification:=marqueurTemps
Fin de cas
```

Une fois que cela est implémenté dans votre base, il suffit d'écrire la méthode projet ***CREER DOCUMENTATION*** listée ci-dessous. Nous utilisons **PROPRIETES DOCUMENT** et **CHANGER PROPRIETES DOCUMENT** pour gérer la date et l'heure de création et de modification des documents.

```
 //Méthode projet CREER DOCUMENTATION

C_ALPHA(255;$vsPath;$vsDocPathName;$vsDocName)
C_ENTIER LONG($vlDoc)
C_BOOLEEN($vbOnWindows;$vbDoIt;$vbLocked;$vbInvisible)
C_HEURE($vhDocRef;$vhCreatedAt;$vhModifiedAt)
C_DATE($vdCreatedOn;$vdModifiedOn)

Si(Type application=4D Client)
 // Si 4D Client est utilisé, sauvegarder les documents localement
 // c'est-à-dire sur le poste client où se trouve 4D Client
   $vsPath:=Nom long vers chemin d'accès(Type application)
Sinon
 // Sinon, sauvegarder les documents là où se trouve le fichier de données
   $vsPath:=Nom long vers chemin d'accès(Fichier donnees)
Fin de si
 // Stocker les documents dans un répertoire nommé arbitrairement "Documentation"
$vsPath:=$vsPath+"Documentation"+Caractere(Symbole séparateur)
 // Si ce répertoire n'existe pas, le créer
Si(Tester chemin acces($vsPath) #Est un dossier)
   CREER DOSSIER($vsPath)
Fin de si
 // Etablir la liste des documents existants
 // car nous allons devoir supprimer ceux qui sont obsolètes, autrement dit
 // ceux dont les enregistrements correspondants ont été supprimés.
TABLEAU ALPHA(255;$asDocument;0)
LISTE DES DOCUMENTS($vsPath;$asDocument)
 // Sélection de tous les enregistrements de la table [Documents]
TOUT SELECTIONNER([Documents])
 // Pour chaque enregistremnt
$vlNbRecords:=Enregistrements trouves([Documents])
$vlNbDocs:=0
$vbOnWindows:=Sous Windows
Boucle($vlDoc;1;$vlNbRecords)
 // Supposons que nous aurons à (re)créer le document sur disque
   $vbDoIt:=Vrai
 // Calcul du nom et du chemin d'accès au document
   $vsDocName:="DOC"+Chaine([Documents]Numéro;"00000")
   $vsDocPathName:=$vsPath+$vsDocName
 // Est-ce que ce document existe déjà ?
   Si(Tester chemin acces($vsDocPathName+".HTM")=Est un document)
 // Si oui, retirer le document de la liste des documents
 // qui peuvent être supprimés
      $vlElem:=Chercher dans tableau($asDocument;$vsDocName+".HTM")
      Si($vlElem>0)
         SUPPRIMER DANS TABLEAU($asDocument;$vlElem)
      Fin de si
 // Est-ce que le document a été stocké après la dernière modification de l'enregistrement?
      PROPRIETES DOCUMENT($vsDocPathName+".HTM";$vbLocked;$vbInvisible;$vdCreatedOn;
      $vhCreatedAt;$vdModifiedOn;$vhModifiedAt)
      Si(marqueurTemps($vdModifiedOn;$vhModifiedAt)>=[Documents]Marqueur modification)
 //Si oui, nous n'avons pas besoin de recréer le document
         $vbDoIt:=Faux
      Fin de si
```

```
    Sinon
  //Le document n'existe pas, mettre ces deux variables à zéro, pour que
  // nous sachions que nous devrons les traiter avant de fixer les propriétés finales
  // du document
      $vdModifiedOn:=!00/00/00!
      $vhModifiedAt:=†00:00:00†
    Fin de si
  // Avons-nous besoin de (re)créer le document?
    Si($vbDoIt)
  // Si oui, incrémenter le nombre de documents mis à jour
      $vlNbDocs:=$vlNbDocs+1
  // Supprimer le document s'il existe déjà
      SUPPRIMER DOCUMENT($vsDocPathName+".HTM")
  // Et le recréer
      Si($vbOnWindows)
         $vhDocRef:=Creer document($vsDocPathName;"HTM")
      Sinon
         $vhDocRef:=Creer document($vsDocPathName+".HTM")
      Fin de si
      Si(OK=1)
  //...
  // Ecrivons ici le contenu du document
  // ...
         FERMER DOCUMENT($vhDocRef)
         Si($vdModifiedOn=!00/00/00!)
  // Le document n'existait pas, fixer les valeurs correctes pour
  // la date et l'heure de modification
            $vdModifiedOn:=Date du jour
            $vhModifiedAt:=Heure courante
         Fin de si
  // Changer les propriétés du document de telle manière que sa date et son heure de création
  // soit égales à celles de l'enregistrement correspondant
         CHANGER PROPRIETES DOCUMENT($vsDocPathName+".HTM";$vbLocked;$vbInvisible;Marqueur vers
date([Documents]Marqueur création);Marqueur vers heure([Documents]Marqueur création);$vdModifiedOn;$vhModifiedAt)
      Fin de si
    Fin de si
  // Juste pour savoir ce qui se passe
    CHANGER TITRE FENETRE("Traitement du document "+Chaine($vlDoc)+" sur "+Chaine($vlNbRecords))
    ENREGISTREMENT SUIVANT([Documents])
 Fin de boucle
  //Suppression des documents obsolètes, c'est-à-dire ceux
  // qui sont toujours dans le tableau $asDocument
 Boucle($vlDoc;1;Taille tableau($asDocument))
    SUPPRIMER DOCUMENT($vsPath+$asDocument{$vlDoc})
    CHANGER TITRE FENETRE("Suppression du document obsolète: "+Caractere(34)+$asDocument{$vlDoc}+Caractere(34))
 Fin de boucle
  //C'est la fin
 ALERTE("Nombre de documents traités : "+Chaine($vlNbRecords)+Caractere(13)+"Nombre de documents mis à jour :
"+Chaine($vlNbDocs)+Caractere(13)+"Nombre de documents supprimés : "+Chaine(Taille tableau($asDocument)))
```

PROPRIETES DU VOLUME

PROPRIETES DU VOLUME (volume ; taille ; utilisé ; libre)

Paramètre	Type		Description
volume	Chaîne	➡	Nom du volume
taille	Réel	⬅	Taille du volume exprimée en octets
utilisé	Réel	⬅	Place utilisée sur le volume exprimée en octets
libre	Réel	⬅	Place libre sur le volume exprimée en octets

Description

La commande **PROPRIETES DU VOLUME** retourne la taille, la place utilisée et la place libre sur le volume dont le nom est passé dans *volume*. Ces valeurs sont exprimées en octets.

Note : Si *volume* indique un volume distant non monté, la variable OK prend la valeur 0 et les trois paramètres retournent -1.

Exemple

Votre application comprend des opérations par lots qui sont exécutées la nuit ou pendant le week-end. Ces opérations stockent des fichiers temporaires sur disque. Pour que cette méthode soit aussi autonome et souple que possible, vous écrivez une routine qui va automatiquement chercher et utiliser le premier volume ayant de la place disponible pour les fichiers temporaires. Voici la méthode :

```
  ` Méthode projet Chercher volume pour place
  ` Chercher volume pour place ( Reel ) -> Alpha
  ` Chercher volume pour place ( Place nécessaire en octets ) -> Nom du volume ou chaîne vide

C_ALPHA(31;$0)
C_ALPHA(255;$vaNomDoc)
C_ENTIER LONG($vlNbVolumes;$vlVolume)
C_REEL($1;$vlTaille;$vlUtilisé;$vlLibre)
C_HEURE($vhDocRef)

  ` Initialiser le résultat de la fonction
$0:=""
  ` Protéger toutes les opérations d'entrée/sortie par une méthode d'interruption d'erreur
APPELER SUR ERREUR("METHODE ERREUR")
  ` Obtenir la liste des volumes
TABLEAU ALPHA(31;$taVolumes;0)
gErreur:=0
LISTE DES VOLUMES($taVolumes)
Si(gErreur=0)
  ` Si nous sommes sous Windows, ignorer les deux lecteurs de disquettes
   Si(Sous Windows)
      $vlVolume:=Chercher dans tableau($taVolumes;"A:\\")
      Si($vlVolume>0)
         SUPPRIMER DANS TABLEAU($taVolumes;$vlVolume)
      Fin de si
      $vlVolume:=Chercher dans tableau($taVolumes;"B:\\")
      Si($vlVolume>0)
         SUPPRIMER DANS TABLEAU($taVolumes;$vlVolume)
      Fin de si
   Fin de si
   $vlNbVolumes:=Taille tableau($taVolumes)
  ` Pour chaque volume
   Boucle($vlVolume;1;$vlNbVolumes)
  ` Obtenir la taille, la place utilisée et la place libre
      gErreur:=0
      PROPRIETES DU VOLUME($taVolumes{$vlVolume};$vlTaille;$vlUtilisé;$vlLibre)
      Si(gErreur=0)
  ` Est-ce que la place libre est suffisante (plus 32K) ?
         Si($vlLibre>=($1+32768))
  ` Si oui, vérifier que le volume n'est pas verrouillé...
            $vaNomDoc:=$taVolumes{$vlVolume}+Caractere(Symbole séparateur)+"XYZ"+Chaine(Hasard)+".TXT"
            $vhDocRef:=Creer document($vaNomDoc)
            Si(OK=1)
               FERMER DOCUMENT($vhDocRef)
               SUPPRIMER DOCUMENT($vaNomDoc)
  ` Si tout est ok, retourner le nom du volume
               $0:=$taVolumes{$vlVolume}
               $vlVolume:=$vlNbVolumes+1
            Fin de si
         Fin de si
      Fin de si
   Fin de boucle
Fin de si
APPELER SUR ERREUR("")
```

cette méthode projet est ajoutée à votre application, vous pouvez écrire :

```
$vaVolume:=Chercher volume pour place(100*1024*1024)
Si($vaVolume#"")
  ` Continuer
Sinon
   ALERTE("Un volume avec au moins 100 Mo d'espace libre est nécessaire !")
Fin de si
```

RESOUDRE ALIAS

RESOUDRE ALIAS (cheminAlias ; cheminCible)

Paramètre	Type		Description
cheminAlias	Chaîne	⇨	Nom ou chemin d'accès complet de l'alias/ du raccourci
cheminCible	Chaîne	⇦	Nom ou chemin d'accès complet de la cible de l'alias/du raccourci

Description

La commande **RESOUDRE ALIAS** retourne le chemin d'accès complet du fichier ou dossier cible d'un alias (appelé "raccourci" sous Windows).

Vous passez dans *cheminAlias* le nom ou le chemin d'accès complet de l'alias.

Note : Sous Windows, les raccourcis sont des fichiers dont l'extension est ".LNK". Si vous ne passez pas cette extension, la commande l'ajoute automatiquement.

Après l'exécution de la commande, la variable *cheminCible* contient le chemin d'accès complet du fichier ou dossier cible de l'alias et la variable système OK prend la valeur 1.
Si le chemin passé dans *cheminAlias* correspond à un fichier et non à un alias, *cheminCible* retourne le chemin d'accès du fichier et la variable système OK prend la valeur 0.

Variables et ensembles système

Si *cheminAlias* désigne bien un alias/raccourci, la variable système OK prend la valeur 1. Si *cheminAlias* désigne un fichier standard, la variable système OK prend la valeur 0.

Selectionner document

Selectionner document (répertoire ; typesFichiers ; titre ; options {; sélectionnés}) -> Résultat

Paramètre	Type		Description
répertoire	Texte, Entier long	➡	• Chemin d'accès du répertoire à afficher par défaut dans la boîte de dialogue de sélection, ou • Chaîne vide pour afficher le dossier d'utilisateur par défaut ("Mes documents" sous Windows, "Documents" sous Mac OS), ou • Numéro de chemin d'accès mémorisé
typesFichiers	Texte	➡	Liste des types de documents à filtrer, ou "*" pour ne pas filtrer les documents
titre	Texte	➡	Titre de la boîte de dialogue de sélection
options	Entier long	➡	Option(s) de sélection
sélectionnés	Tableau texte	⬅	Tableau contenant la liste des chemins d'accès + les noms des fichiers sélectionnés
Résultat	Chaîne	➲	Nom du fichier sélectionné (premier fichier de la liste en cas de sélection multiple)

Description

La commande **Selectionner document** affiche une boîte de dialogue standard d'ouverture de documents, permettant à l'utilisateur de désigner un ou plusieurs fichier(s), et retourne le nom et/ou le chemin d'accès complet du ou des fichier(s) sélectionné(s).

Le paramètre *répertoire* indique le dossier dont le contenu doit être affiché initialement dans la boîte de dialogue d'ouverture de documents. Vous pouvez passer trois types de valeurs :

- un texte contenant le chemin d'accès complet du répertoire à afficher.
- une chaîne vide ("") pour afficher le dossier d'utilisateur par défaut du système ("Mes documents" sous Windows, "Documents" sous Mac OS).
- un numéro de chemin d'accès mémorisé (de 1 à 32000) pour afficher le dossier associé.
  Avec ce principe, vous pouvez conserver en mémoire le chemin d'accès du dossier ouvert au moment où l'utilisateur a cliqué sur le bouton de sélection, c'est-à-dire le dossier choisi par l'utilisateur. Lors du premier appel d'un numéro arbitraire (par exemple 5), la commande affiche le dossier d'utilisateur par défaut du système (équivaut à passer une chaîne vide). L'utilisateur pourra alors éventuellement naviguer parmi les dossiers de son disque dur. Au moment où il cliquera sur le bouton de sélection, le chemin d'accès sera mémorisé et associé au numéro 5. Lors des appels suivants du numéro 5, le chemin d'accès mémorisé sera utilisé par défaut. En cas de sélection d'un nouvel emplacement, le chemin numéro 5 sera mis à jour, et ainsi de suite.
  Ce mécanisme vous permet de mémoriser jusqu'à 32000 chemins d'accès. Sous Windows, chaque chemin est conservé durant la session uniquement. Sous Mac OS, les chemins sont conservés par le système, ils restent mémorisés d'une session à l'autre.

Note : Ce mécanisme est identique à celui utilisé par la commande **Selectionner dossier**. Les numéros de chemins d'accès mémorisés sont partagés entre les deux commandes.

Passez dans le paramètre *typeFichiers* le ou les type(s) de fichier(s) pouvant être sélectionnés dans la boîte de dialogue d'ouverture. Vous pouvez passer une liste de plusieurs types séparés par un *;* (point virgule). Pour chaque type défini, une ligne sera ajoutée dans le menu de choix de type de la boîte de dialogue.

- Sous Mac OS, vous pouvez passer soit un type Mac OS classique (TEXT, APPL, etc.), soit un type UTI (Uniform Type Identifier). Les types UTIs ont été définis par Apple afin de répondre aux besoins d'uniformisation des types de fichiers. Par exemple, "public.text" est le type UTI des fichiers de type texte. Pour plus d'informations sur les UTIs, reportez-vous à l'adresse https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/understanding_utis/understand_utis_conc/understand_utis_conc.html (documentation en anglais).
- Sous Windows, vous pouvez également passer un type de fichier classique Mac OS — 4D effectue la correspondance en interne — ou l'extension des fichiers (.txt, .exe, etc.). A noter que sous Windows, l'utilisateur aura la possibilité de "forcer" l'affichage de tous les types de fichiers en saisissant *.* dans la boîte de dialogue. Toutefois dans ce cas, 4D effectuera une vérification supplémentaire des types des fichiers sélectionnés : si l'utilisateur sélectionne un type de fichier non autorisé, la commande retourne une erreur.

Si vous ne souhaitez pas restreindre les fichiers affichés à un ou plusieurs types, passez la chaîne "*" (étoile) ou ".*" dans *typeFichiers*.

Passez dans le paramètre *titre* le libellé devant apparaître dans la boîte de dialogue. Par défaut, si vous passez une chaîne vide, le libellé "Ouvrir" est affiché.

Le paramètre *options* permet de spécifier les fonctions avancées autorisées dans la boîte de dialogue d'ouverture. 4D vous propose les constantes prédéfinies suivantes, placées dans le thème **Documents système**. Vous pouvez passer une constante ou une combinaison de constantes.

Constante	Type	Valeur	Comment
Fichiers multiples	Entier long	1	Autorise la sélection simultanée de plusieurs fichiers à l'aide des combinaisons **Maj+clic** (sélection contiguë) et **Ctrl+clic** (Windows) ou **Commande+clic** (Mac OS). Dans ce cas, le paramètre *sélectionnés*, s'il est passé, contient la liste de tous les fichiers sélectionnés. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas la sélection de plusieurs fichiers.
Ouverture progiciel	Entier long	2	(Mac OS uniquement) Autorise l'ouverture des progiciels (packages) en tant que dossiers et donc la visualisation/sélection de leur contenu. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas l'ouverture des progiciels.
Saisie nom fichier	Entier long	32	Permet à l'utilisateur à saisir un nom de fichier dans une boîte de dialogue de sauvegarde. Aucun fichier n'est sauvegardé, il revient au développeur de créer un fichier en réponse à cette action (la variable système Document est mise à jour). Dans ce contexte, il est possible de passer un chemin de fichier dans le paramètre *répertoire*. Le nom du fichier sera suggéré dans la boîte de dialogue de sauvegarde et son répertoire parent sera utilisé comme chemin par défaut.
Sélection alias	Entier long	8	Autorise la sélection de raccourcis (Windows) ou d'alias (Mac OS) en tant que documents. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas la sélection de raccourcis ou d'alias en tant que tels. Si l'utilisateur sélectionne ce type de document, la commande retourne le chemin de l'élément cible. Lorsque vous passez la constante, la commande retourne le chemin de l'alias ou du raccourci lui-même.
Sélection progiciel	Entier long	4	(Mac OS uniquement) Autorise la sélection de progiciels (packages) en tant qu'entités. Par défaut, si cette constante n'est pas utilisée, la commande ne permet pas de sélectionner les progiciels en tant que tels.
Utiliser fenêtre feuille	Entier long	16	(Mac OS uniquement) Affiche la boîte de dialogue de sélection sous forme de fenêtre feuille (cette option est ignorée sous Windows). Les fenêtres feuilles sont des fenêtres spécifiques de l'interface Mac OS X, bénéficiant notamment d'une animation graphique (pour plus d'informations, reportez-vous à la section **Types de fenêtres (compatibilité)**). Par défaut, si cette constante n'est pas utilisée, la commande affiche une boîte de dialogue standard.

Si vous ne souhaitez pas utiliser d'option, passez 0 dans le paramètre *options*.

Le paramètre facultatif *sélectionnés* permet de récupérer le chemin d'accès complet (chemin d'accès + nom) de chaque fichier sélectionné par l'utilisateur. La commande crée, dimensionne et remplit le tableau en fonction de la sélection de l'utilisateur. Ce paramètre est utile lorsque l'option Fichiers multiples est utilisée, ou lorsque vous souhaitez connaître le chemin d'accès du fichier sélectionné (il suffit dans ce cas de soustraire de la valeur du tableau le nom du fichier retourné par la commande). Si aucun fichier n'a été sélectionné, le tableau est retourné vide.

Note : Sous Mac OS, un progiciel sélectionné est considéré comme un dossier. Le chemin d'accès retourné dans le tableau *sélectionnés* comporte un caractère ":" final. Par exemple : **Disque:Applications:4D:4D v11.4:FR:4D Volume Desktop.app:**

La commande retourne le nom (nom+extension sous Windows) du fichier sélectionné. Si plusieurs fichiers ont été sélectionnés, la commande retourne le nom du premier fichier de la liste des fichiers sélectionnés. La liste des fichiers peut être récupérée dans le paramètre *sélectionnés*. Si aucun fichier n'a été sélectionné, la commande retourne une chaîne vide.

Exemple 1

Cet exemple permet de désigner un fichier de données 4D :

```
C_ENTIER LONG($platForm)
PROPRIETES PLATE FORME($platForm)
Si($platForm=Windows)
   $DocType:=".4DD"
Sinon
   $DocType:="com.4d.4d.data-file" `Type UTI
Fin de si
$Options:=Sélection alias+Ouverture progiciel+Utiliser fenêtre feuille
$Doc:=Selectionner document("";$DocType;"Sélectionner le fichier de données";$Options)
```

Exemple 2

Création d'un document personnalisé par l'utilisateur :

```
$doc:=Selectionner document(Dossier systeme(Dossier documents)+"Report.pdf";"pdf";"Nom de l'état :";Saisie nom fichier)
Si(OK=1)
   BLOB VERS DOCUMENT(Document;$blob) // $blob contient le document à enregistrer
Fin de si
```

Variables et ensembles système

Si la commande a été correctement exécutée et qu'un document valide a été sélectionné, la variable système OK prend la valeur 1 et la variable système Document contient le chemin d'accès complet du fichier sélectionné.
Si aucun fichier n'a été sélectionné (par exemple si l'utilisateur a cliqué sur le bouton **Annuler** dans la boîte de dialogue d'ouverture), la variable système OK prend la valeur 0 et la variable système Document est vide.

Selectionner dossier

Selectionner dossier ({message }{;}{ répertoire {; options}}) -> Résultat

Paramètre	Type		Description
message	Chaîne	➡	Titre de la fenêtre de sélection
répertoire	Chaîne, Entier long	➡	Chemin d'accès du répertoire par défaut ou Chaîne vide pour afficher le dossier d'utilisateur par défaut ("Mes documents" sous Windows, "Documents" sous Mac OS), ou Numéro de chemin d'accès mémorisé
options	Entier long	➡	Option(s) de sélection sous Mac OS
Résultat	Chaîne	↩	Chemin d'accès au dossier sélectionné

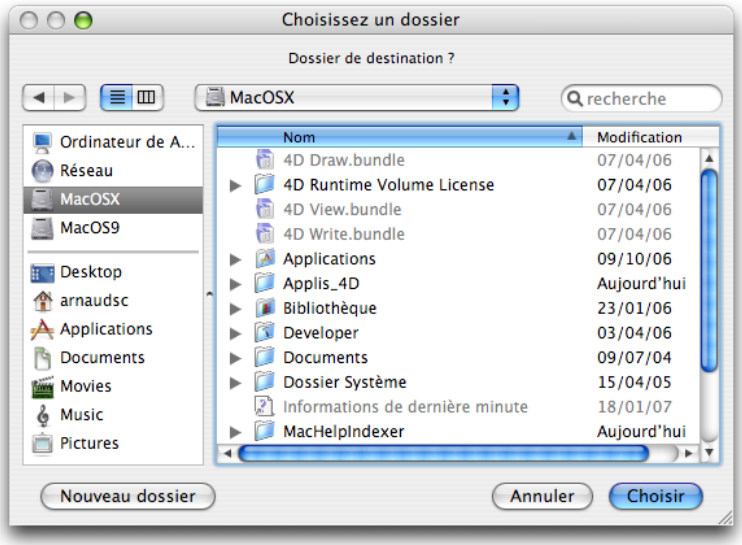
Description

La commande **Selectionner dossier** affiche une boîte de dialogue permettant de désigner manuellement un dossier, et de récupérer en retour de fonction le chemin d'accès complet au dossier sélectionné. Le paramètre facultatif *répertoire* vous permet de désigner un emplacement de dossier qui sera affiché initialement dans la boîte de dialogue de sélection de dossier.

Note : Cette commande ne modifie pas le dossier courant de l'application 4D.

La commande **Selectionner dossier** affiche une boîte de dialogue standard de navigation à travers les volumes et les dossiers du poste.
Le paramètre optionnel *message* permet d'afficher une ligne d'information dans la boîte de dialogue (dans notre exemple, *message* a pour valeur "Dossier de destination ?").
Windows :

Mac OS :



Vous pouvez utiliser le paramètre *répertoire* pour proposer un emplacement de dossier par défaut dans la boîte de dialogue de sélection de dossier. Vous pouvez passer dans ce paramètre trois types de valeurs :

- un chemin d'accès de dossier valide utilisant la syntaxe de la plate-forme courante.
- une chaîne vide ("") pour afficher le dossier d'utilisateur par défaut du système ("Mes documents" sous Windows, "Documents" sous Mac OS).
- un numéro de chemin d'accès mémorisé (de 1 à 32000) pour afficher le dossier associé. Avec ce principe, vous pouvez conserver en mémoire le chemin d'accès du dossier ouvert au moment où l'utilisateur a cliqué sur le bouton de sélection, c'est-à-dire le dossier choisi par l'utilisateur. Lors du premier appel d'un numéro arbitraire (par exemple 5), la commande affiche le dossier d'utilisateur par défaut du système (équivaut à passer une chaîne vide). L'utilisateur peut alors éventuellement naviguer parmi les dossiers de son disque dur. Au moment où il clique sur le bouton de sélection, le chemin d'accès est mémorisé et associé au numéro 5. Lors des appels suivants du numéro 5, le chemin d'accès mémorisé sera utilisé par défaut. En cas de sélection d'un nouvel emplacement, le chemin numéro 5 sera mis à jour, et ainsi de suite.
  Ce mécanisme vous permet de mémoriser jusqu'à 32000 chemins d'accès. Sous Windows, chaque chemin est conservé durant la session uniquement. Sous Mac OS, les chemins restent mémorisés d'une session à l'autre. Si le chemin d'accès est incorrect, le paramètre *cheminDéfaut* est ignoré.

Note : Ce mécanisme est identique à celui utilisé par la commande **Selectionner document**. Les numéros de chemins d'accès mémorisés sont partagés

entre les deux commandes.

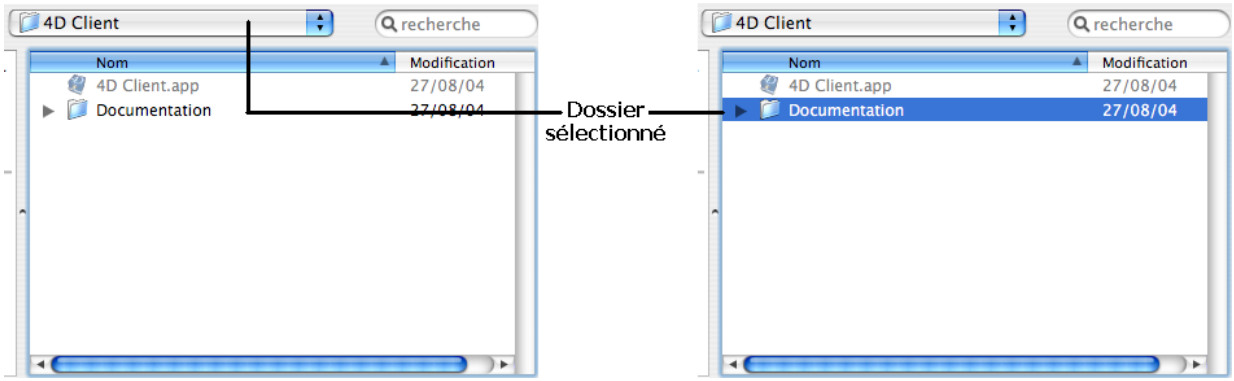
Le paramètre *options* vous permet de bénéficier de fonctions supplémentaires sous Mac OS. Vous pouvez passer dans ce paramètre les constantes suivantes, placées dans le thème **Documents système** :

Constante	Type	Valeur	Comment
Ouverture progiciel	Entier long	2	(Mac OS uniquement) Autorise l'ouverture des progiciels (packages) en tant que dossiers et donc la visualisation/sélection de leur contenu. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas l'ouverture des progiciels.
Utiliser fenêtre feuille	Entier long	16	(Mac OS uniquement) Affiche la boîte de dialogue de sélection sous forme de fenêtre feuille (cette option est ignorée sous Windows). Les fenêtres feuilles sont des fenêtres spécifiques de l'interface Mac OS X, bénéficiant notamment d'une animation graphique (pour plus d'informations, reportez-vous à la section **VISUALISER SELECTION**). Par défaut, si cette constante n'est pas utilisée, la commande affiche une boîte de dialogue standard.

Vous pouvez passer une constante ou la combinaison des deux. Ces options sont prises en compte sous Mac OS uniquement. Sous Windows, le paramètre *options* est ignoré s'il est passé.
L'utilisateur sélectionne un dossier en cliquant sur le bouton **OK** (Windows) ou **Sélectionner** (Mac OS). Le chemin d'accès au dossier choisi est alors retourné par la fonction.

- Sous Windows, la chaîne retournée est du type "C:\Dossier1\Dossier2\DossierSélectionné\"
- Sous Mac OS, la chaîne retournée est du type "Disque:Dossier1:Dossier2:DossierSélectionné:"

Note : Sous Mac OS, selon que le nom du dossier est sélectionné ou non dans la boîte de dialogue, le chemin retourné est différent.



4D Server : Cette fonction permet de visualiser les volumes connectés aux postes clients. Il n'est pas possible de l'appeler depuis une procédure stockée.

Si l'utilisateur clique sur le bouton de sélection, la variable système OK prend la valeur 1. Si l'utilisateur clique sur le bouton d'annulation, OK prend la valeur 0 et la fonction retourne une chaîne vide.

Note : Sous Windows, si l'utilisateur a sélectionné certains éléments incorrects tels que "Poste de travail", "Corbeille", etc., la variable système OK prend la valeur 0, même si la boîte de dialogue est validée.

Exemple

L'exemple suivant permet de sélectionner le dossier dans lequel toutes les images de la bibliothèque d'images seront enregistrées :

```
$DossierImages:=Selectionner dossier("Sélectionnez un dossier pour vos images.")
LISTE IMAGES DANS BIBLIOTHEQUE(pictRefs;pictNoms)
Boucle($n;1;Taille tableau(pictNoms))
   LIRE IMAGE DANS BIBLIOTHEQUE(pictRefs{$n};$vPictSauvegarde)
   ECRIRE FICHIER IMAGE($DossierImages+pictNoms{$n};$vPictSauvegarde)
Fin de boucle
```

SUPPRIMER DOCUMENT

SUPPRIMER DOCUMENT (nomFichier)

Paramètre	Type		Description
nomFichier	Chaîne	⇨	Nom de document ou Chemin d'accès complet au document

Description

SUPPRIMER DOCUMENT supprime le document dont vous avez passé le nom dans *document*.

Si le nom du document ou le chemin d'accès saisi est incorrect, une erreur est générée. Il en va de même si vous tentez de supprimer un document ouvert.

SUPPRIMER DOCUMENT n'accepte pas de chaîne vide dans le paramètre *document*. Si vous passez une chaîne vide, la boîte de dialogue standard d'ouverture de fichiers ne s'affiche pas et une erreur est générée.

Attention : SUPPRIMER DOCUMENT peut supprimer tout fichier disque, y compris des fichiers créés par d'autres applications ou les applications elles-mêmes. La commande **SUPPRIMER DOCUMENT** doit donc être utilisée avec précaution. La suppression d'un document est une opération définitive et irréversible.

Exemple 1

L'exemple suivant supprime le document appelé Note :

```
SUPPRIMER DOCUMENT("Note") ` Suppression du document
```

Exemple 2

Reportez-vous à l'exemple de la commande **AJOUTER DONNEES AU CONTENEUR**.

Variables et ensembles système

La suppression d'un document met la variable système OK à 1. Si **SUPPRIMER DOCUMENT** ne peut pas supprimer le document, la variable système OK prend la valeur 0.

SUPPRIMER DOSSIER

SUPPRIMER DOSSIER (dossier {; optionSuppression})

Paramètre	Type		Description
dossier	Chaîne	⇒	Nom ou chemin d'accès complet du dossier à supprimer
optionSuppression	Entier long	⇒	Option de suppression du dossier

Description

La commande **SUPPRIMER DOSSIER** supprime le dossier dont vous avez passé le nom ou le chemin d'accès complet dans *dossier*.

Par défaut pour des raisons de sécurité, si vous omettez le paramètre *optionSuppression*, **SUPPRIMER DOSSIER** permet uniquement la suppression de dossiers vides. Si vous souhaitez que cette commande puisse supprimer des dossiers non vides, vous devez utiliser le paramètre *optionSuppression*. Vous pouvez passer dans ce paramètre l'une des constantes suivantes, placées dans le thème "**Documents système**" :

Constante	Type	Valeur	Comment
Supprimer avec contenu	Entier long	1	Supprime le dossier ainsi que son éventuel contenu
Supprimer si vide	Entier long	0	Supprime le dossier uniquement s'il est vide

- Si vous passez Supprimer si vide ou omettez le paramètre *optionSuppression* :
  - Le dossier désigné par le paramètre *dossier* n'est supprimé que s'il est vide ; sinon, la commande ne fait rien et une erreur -47 (Fichier déjà ouvert, ou dossier non vide) est générée.
  - Si le dossier désigné n'existe pas, l'erreur -120 (Tentative d'accès à un fichier avec un chemin d'accès spécifiant un répertoire inexistant) est générée.

- Si vous passez Supprimer avec contenu :
  - Le dossier ainsi que tout son contenu sont supprimés.
    Attention : Si le dossier est verrouillé ou en lecture seule, il sera néanmoins supprimé si l'utilisateur courant dispose des droits d'accès nécessaires.
  - Si le dossier désigné ou un des fichiers qu'il contient ne peut pas être supprimé, la procédure de suppression est abandonnée dès que le premier élément inaccessible est atteint, et une erreur(*) est retournée. Dans ce cas, le dossier ne sera que partiellement supprimé. Il est cependant possible d'utiliser la commande **LIRE PILE DERNIERE ERREUR** pour obtenir le nom et le chemin d'accès du fichier à l'origine de l'erreur.
  - Si le dossier désigné n'existe pas, la commande ne fait rien et aucune erreur n'est générée.

    (*) sous Windows : -54 (Tentative d'écriture dans un fichier verrouillé)
    sous OS X : -45 (Fichier verrouillé ou chemin d'accès invalide)

Vous pouvez intercepter ces erreurs à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR** .

Taille document

Taille document (document {; *}) -> Résultat

Paramètre	Type		Description
document	Chaîne, RefDoc	⇒	Numéro de référence de document ou Nom de document
*	Opérateur	⇒	(Mac OS uniquement) Si omis : taille de la data fork, si passé : taille de la resource fork
Résultat	Réel	⇐	Taille (en octets) de document

Description

La commande **Taille document** retourne la taille, exprimée en octets, d'un document.

Si le document est ouvert, passez son numéro de référence dans *document*.
Si le document n'est pas ouvert, passez son nom ou son chemin d'accès dans *document*.

Sous Mac OS, si vous ne passez pas le paramètre optionnel *, la taille de la data fork est retournée. Si vous passez le paramètre optionnel *, la taille de la resource fork est retournée.

Tester chemin acces

Tester chemin acces (cheminAccès) -> Résultat

Paramètre	Type		Description
cheminAccès	Chaîne	➡	Chemin d'accès à un dossier ou un document
Résultat	Entier long	⮌	1= cheminAccès est un document existant 0 = cheminAccès est un dossier existant <0 = chemin d'accès invalide, code d'erreur du gestionnaire de fichiers du système

Description

La fonction **Tester chemin acces** vérifie si le document ou le dossier dont vous avez passé le chemin d'accès et le nom dans *cheminAccès* est présent sur le disque. Vous pouvez passer un chemin d'accès relatif ou absolu, exprimé dans la syntaxe du système courant.

Si un document est trouvé, **Tester chemin acces** retourne 1. Si un dossier est trouvé, **Tester chemin acces** retourne 0.

4D propose les constantes prédéfinies suivantes :

Constante	Type	Valeur
Est un document	Entier long	1
Est un dossier	Entier long	0

Si aucun document ou dossier n'est trouvé, **Tester chemin acces** retourne une valeur négative (par exemple -43 pour "Fichier non trouvé").

Exemple

L'exemple suivant teste la présence du document “Journal” dans le dossier de la base et le crée s'il n'existe pas :

```
Si(Tester chemin acces("Journal")#Est un document)
   $vhDocRef:=Creer document("Journal")
   Si(OK=1)
      FERMER DOCUMENT($vhDocRef)
   Fin de si
Fin de si
```

TEXTE VERS DOCUMENT

TEXTE VERS DOCUMENT (nomFichier ; texte {; jeuCaractères {; modeRetour}})

Paramètre	Type		Description
nomFichier	Chaîne	➡	Nom de document ou Chemin d'accès à un document
texte	Texte	➡	Texte à stocker dans un document
jeuCaractères	Texte, Entier long	➡	Nom ou Numéro de jeu de caractères
modeRetour	Entier long	➡	Mode de traitement des retours à la ligne

Description

La commande **TEXTE VERS DOCUMENT** permet d'écrire directement le *texte* dans un fichier sur disque.

Passez dans *nomFichier* le nom ou le chemin d'accès du fichier à écrire. Si le fichier n'existe pas, il est créé. S'il existe déjà sur le disque, son contenu précédent est écrasé. Si le fichier existe mais est déjà ouvert, son contenu est verrouillé et une erreur est générée. Vous pouvez passer dans *nomFichier* :

- uniquement le nom du fichier, par exemple "monFichier.txt" : dans ce cas, le fichier sera placé à côté du fichier de structure de l'application.
- un chemin d'accès relatif au fichier de structure de l'application, par exemple "\\docs\\monFichier.txt" sous Windows ou ":docs:monFichier.txt" sous OS X.
- un chemin d'accès absolu, par exemple "c:\\app\\docs\\monFichier.txt" sous Windows ou "MacHD:docs:monFichier.txt" sous OS X.

Si vous souhaitez permettre à l'utilisateur de désigner le nom ou l'emplacement du document, utilisez les commandes **Ouvrir document** ou **Creer document** ainsi que la variable système **Document**.

Note : Par défaut, les documents générés par cette commande n'ont pas d'extension. Vous devez passer une extension dans *nomFichier*. Vous pouvez également utiliser la commande **CHANGER TYPE DOCUMENT**.

Passez dans le paramètre *texte* le texte à écrire sur disque. Il peut s'agir d'une constante littérale ("mon texte"), d'un champ texte ou d'une variable texte 4D.

Vous pouvez passer dans *jeuCaractères* le jeu de caractères à utiliser pour l'écriture du document. Vous pouvez passer une chaîne contenant le nom standard du jeu (par exemple "ISO-8859-1" ou "UTF-8") ou son identifiant MIBEnum (entier long). Pour plus d'informations sur la liste des jeux de caractères pris en charge par 4D, reportez-vous à la description de la commande **CONVERTIR DEPUIS TEXTE**. Si une BOM (Byte Order Mark) existe pour le jeu de caractères, 4D l'insère dans le document. Si vous ne précisez pas de jeu de caractères, 4D utilise par défaut le jeu de caractères "UTF_8" et une BOM.

Vous pouvez passer dans *modeRetour* un entier long indiquant le traitement à effectuer sur les caractères de fin de ligne avant de les stocker dans le fichier. Vous pouvez passer une des constantes suivantes, placées dans le thème "**Documents système**" :

Constante	Type	Valeur	Comment
Document avec CR	Entier long	3	Les fins de ligne sont convertis au format OS X : CR (*carriage return*)
Document avec CRLF	Entier long	2	Les fins de ligne sont convertis au format Windows : CRLF (*carriage return* + *line feed*)
Document avec format natif	Entier long	1	(Défaut) Les fins de ligne sont convertis au format natif de la plate-forme d'exécution : CR (*carriage return*) sous OS X, CRLF (*carriage return* + *line feed*) sous Windows
Document avec LF	Entier long	4	Les fins de ligne sont convertis au format Unix : LF (*line feed*)
Document inchangé	Entier long	0	Aucun traitement

Par défaut, si le paramètre *modeRetour* est omis, les caractères de fin de ligne sont traités en mode natif (1).

Note : Cette commande ne modifie pas la variable OK. En cas d'échec, une erreur est générée, que vous pouvez intercepter à l'aide d'une méthode installées par la commande **APPELER SUR ERREUR**.

Exemple 1

Voici des exemples type d'utilisation de cette commande :

```
TEXTE VERS DOCUMENT("monTest.txt";"Ceci est un test")
TEXTE VERS DOCUMENT("monTest.xml";"Ceci est un test")
```

Exemple 2

Exemple permettant à l'utilisateur de désigner l'emplacement du fichier à créer :

```
$MyTextVar:="Ceci est un test"
APPELER SUR ERREUR("IO ERROR HANDLER")
$vhDocRef :=Creer document("")
    // Stocker le document avec l'extension ".txt"
    // Dans ce cas, l'extension .txt est toujours ajoutée au nom, il n'est pas possible de la modifier
Si(OK=1) // Si le document a bien été créé
      FERMER DOCUMENT($vhDocRef) //Refermer le document
      TEXTE VERS DOCUMENT(Document;$MyTextVar )
       // On écrit le document
Sinon
       // Gestion d'erreur
Fin de si
```

Type document

Type document (nomFichier) -> Résultat

Paramètre	Type		Description
nomFichier	Chaîne	⇨	Nom de document ou Chemin d'accès complet à un document
Résultat	Chaîne	⇦	Extension de fichier Windows (chaîne de 1 à N caractères) ou type de fichier Mac OS (chaîne de 4 caractères)

Description

La commande **Type document** retourne l'extension ou le type du document dont vous avez passé le nom ou le chemin d'accès dans *document*.

Sous Windows, **Type document** retourne l'extension de fichier du document (par exemple *'DOC'* pour un document Microsoft Word, *'EXE'* pour un fichier exécutable, etc.) ou le type de document Mac OS (4 caractères) correspondant si ce dernier a été associé à une extension Windows équivalente par 4D (par exemple *'TEXT'* pour l'extension *'TXT'*) ou par un appel antérieur à la commande **ASSOCIER TYPES FICHIER**.

Sous Mac OS, **Type document** retourne, s'il est défini, le type de fichier Mac OS (4 caractères) du document (par exemple *'TEXT'* pour un document de type Texte, *'APPL'* pour une application double-cliquable, etc.).

Note de compatibilité

L'utilisation des types de fichiers Mac OS est obsolète sous Mac OS X. Comme sous Windows, l'identification des fichiers est désormais basée sur le suffixe de leur nom (cf. section **Présentation des documents système**). Par compatibilité, il reste cependant possible de lire le type Mac OS des documents lorsqu'il a été défini à l'aide de **ASSOCIER TYPES FICHIER**.

Enregistrements

A propos des numéros d'enregistrements

Dans 4D, trois numéros sont associés à un enregistrement :

- Numéro d'enregistrement,
- Numéro dans la sélection,
- Numéro automatique.

Numéro d'enregistrement

Le numéro d'enregistrement est le numéro physique/absolu de l'enregistrement. Ce numéro est automatiquement assigné à chaque nouvel enregistrement et reste le même jusqu'à ce que cet enregistrement soit détruit. Les enregistrements commencent au numéro zéro (0).
Les numéros d'enregistrements ne sont pas uniques car les numéros des enregistrements détruits sont réutilisés pour de nouveaux enregistrements. Ces numéros sont également modifiés lorsque la base est réparée ou compactée.

Numéro dans la sélection

Le numéro dans la sélection est la position de l'enregistrement dans la sélection courante. Ce numéro dépend de la sélection courante. Si la sélection est modifiée ou triée, ce numéro change aussi probablement. La numérotation dans une sélection courante commence à un (1).

Numéro automatique

Le numéro automatique est un numéro unique, non répétable, qui peut être assigné à un champ dans un enregistrement (via la propriété **Incrémentation auto**, l'attribut SQL AUTO_INCREMENT ou la commande **Numerotation automatique**). Il n'est pas automatiquement stocké à chaque enregistrement. Il démarre par défaut à 1 et est incrémenté à chaque création d'un nouvel enregistrement. A la différence des numéros d'enregistrements, un numéro automatique n'est pas réutilisé lorsque l'enregistrement est détruit, ou lorsque la base est compactée ou réparée.
Ces numéros fournissent un moyen d'attribuer un numéro d'identification unique à chaque enregistrement. Si un numéro automatique est incrémenté pendant une transaction, ce numéro n'est pas décrémenté si la transaction est annulée.

Note : 4D n'effectue pas de contrôle lorsque vous modifiez le compteur interne des numéros automatiques d'une table à l'aide de la commande **FIXER PARAMETRE BASE**. Si vous décrémentez ce compteur, les nouveaux enregistrements créés pourront avoir des numéros ayant déjà été attribués.

Exemples de numéros d'enregistrements

Les tableaux suivants comparent le fonctionnement des différents numéros d'enregistrements. Chaque ligne de tableau représente les informations d'un enregistrement. L'ordre des lignes est celui dans lequel les enregistrements seraient affichés dans un formulaire sortie.

- **Colonne des Données** : Les valeurs d'un champ dans chaque enregistrement. Elle contient le nom d'une personne.
- **Colonne de Numéro d'enregistrement (N° Enrg)** : C'est le numéro absolu de l'enregistrement et qui est retourné par la fonction **Numero enregistrement**.
- **Colonne de Numéro dans la sélection (N° Sélection)** : C'est le numéro de position dans la sélection courante, qui est retourné par la fonction **Numero dans selection**.
- **Colonne de Numéro automatique (N° Auto)**: C'est le numéro unique de l'enregistrement, qui est retourné par la fonction **Numerotation automatique**. Ce numéro est stocké dans un champ.

Après saisie des enregistrements

Le premier tableau présente des enregistrements qui viennent d'être saisis.

- L'ordre des enregistrements par défaut est le numéro d'enregistrement.
- Le numéro d'enregistrement commence à 0.
- Le numéro dans la sélection et le numéro automatique commencent à 1.

Données	N° Enrg	N° Sélection	N° Auto
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Sam	3	4	4
Lisa	4	5	5

Note: Les enregistrements restent dans l'ordre par défaut après l'appel de toute commande qui modifie la sélection sans la réordonner, comme par exemple la commande de menu **Tout montrer** en mode Développement ou après l'exécution de la commande **TOUT SELECTIONNER**.

Après un tri des enregistrements

La première partie du tableau présente les enregistrements triés par noms.

- Le numéro d'enregistrement reste associé à l'enregistrement.
- Le numéro dans la sélection reflète la nouvelle position de l'enregistrement dans la sélection triée.
- Le numéro automatique ne change jamais puisqu'il est assigné à la création de chaque enregistrement et stocké avec lui.

Données	N° Enrg	N° Sélection	N° Auto
Lisa	4	1	5
Sabra	2	2	3
Sam	3	3	4
Terri	1	4	2
Tess	0	5	1

Après la suppression d'un enregistrement

Voici le tableau après la destruction de l'enregistrement de Sam.

- Seuls les numéros dans la sélection ont changé (les numéros dans la sélection reflètent l'ordre d'affichage des enregistrements).

Données	N° Enrg	N° Sélection	N° Auto
Lisa	4	1	5
Sabra	2	2	3
Terri	1	3	2
Tess	0	4	1

Après l'ajout d'un enregistrement

Voici le tableau après l'ajout de l'enregistrement Liz.

- Un nouvel enregistrement est ajouté à la fin de la sélection courante.
- Le numéro d'enregistrement de Sam est réutilisé pour le nouvel enregistrement.
- Le numéro automatique a été incrémenté de 1.

Données	N° Enrg	N° Sélection	N° Auto
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Lisa	4	4	5
Liz	3	5	6

Après un changement de sélection et un tri

Le tableau qui suit montre les enregistrements après réduction de la sélection à trois enregistrements qui sont ensuite triés.

- Seul le numéro dans la sélection change.

Données	N° Enrg	N° Sélection	N° Auto
Sabra	2	1	3
Liz	3	2	6
Terri	1	3	2

Utiliser la pile d'enregistrements

Les commandes **EMPILER ENREGISTREMENT** et **DEPILER ENREGISTREMENT** vous permettent de poser (“empiler”) des enregistrements sur le dessus de la pile des enregistrements, et de les enlever (“dépiler”) de la pile.

Chaque process dispose de sa propre pile d'enregistrements pour chaque table. 4D gère pour vous les piles d'enregistrements. Chaque pile d'enregistrements est du type LIFO ("Last-In-First-Out", ce qui peut se traduire par "dernier-entré-premier-sorti"). La capacité de la pile dépend de la mémoire.

Les commandes **EMPILER ENREGISTREMENT** et **DEPILER ENREGISTREMENT** doivent être utilisées avec prudence. Chaque enregistrement empilé utilise une partie de la mémoire disponible. Empiler trop d'enregistrements peut causer l'apparition d'un message du type "mémoire insuffisante" ou une pile pleine.

4D efface de la pile les enregistrements "dépilés" quand vous retournez au menu à la fin de l'exécution de la méthode.

EMPILER ENREGISTREMENT et **DEPILER ENREGISTREMENT** sont utiles lorsque par exemple, en cours de saisie, vous voulez examiner des enregistrements se trouvant dans la même table que celle que vous êtes en train d'utiliser. Pour cela, vous empilez votre enregistrement, cherchez et examinez les enregistrements dans la table (vous copiez des champs dans des variables, par exemple), et finalement vous dépilez l'enregistrement pour le restaurer.

Note pour les utilisateurs de la version 5 de 4D : Quand vous saisissez un enregistrement, si vous devez vérifier l'unicité d'une valeur sur plusieurs champs, utilisez la nouvelle commande **FIXER DESTINATION RECHERCHE**. Cela vous évitera les appels à **EMPILER ENREGISTREMENT** et **DEPILER ENREGISTREMENT** que vous deviez effectuer avant d'utiliser **CHERCHER**, afin de préserver les données saisies dans l'enregistrement courant. **FIXER DESTINATION RECHERCHE** permet d'exécuter une recherche qui ne change pas la sélection ni l'enregistrement courants.

AFFICHER ENREGISTREMENT

AFFICHER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle afficher l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

AFFICHER ENREGISTREMENT affiche l'enregistrement courant de *laTable* dans le formulaire entrée courant. L'enregistrement reste affiché jusqu'à ce qu'un événement provoque un redessinement de la fenêtre. Cet événement peut être l'exécution d'un **AJOUTER ENREGISTREMENT**, le retour au formulaire entrée ou à la barre de menus. **AFFICHER ENREGISTREMENT** ne fait rien s'il n'y a pas d'enregistrement courant.

AFFICHER ENREGISTREMENT est souvent utilisé pour afficher des messages de progression personnalisés. Cette commande peut également servir à générer un "slide show" automatique.

Si une méthode formulaire existe, un événement Sur chargement est généré.

Exemple

L'exemple suivant affiche une série d'enregistrements sous forme de slide show :

```
TOUT SELECTIONNER([Démo]) ` Sélection de tous les enregistrements
FORM FIXER ENTREE([Démo];"Affichage") ` Désignation du formulaire à utiliser
Boucle($i;1;Enregistrements trouves([Démo])) ` Boucle sur tous les enregistrements
   AFFICHER ENREGISTREMENT([Démo]) ` Afficher un enregistrement
   ENDORMIR PROCESS(Numero du process courant;180) ` 3 secondes de pause
   ENREGISTREMENT SUIVANT([Démo]) ` Passage à l'enregistrement suivant
Fin de boucle
```

ALLER A ENREGISTREMENT

ALLER A ENREGISTREMENT ({laTable ;} enregistrement)

Paramètre	Type		Description
laTable	Table	⇨	Table de l'enregistrement de destination ou Table par défaut si ce paramètre est omis
enregistrement	Entier long	⇨	Numéro renvoyé par Numero enregistrement

Description

ALLER A ENREGISTREMENT sélectionne l'enregistrement courant de *table*. Le paramètre *enregistrement* est le numéro renvoyé par la fonction **Numero enregistrement**. Après l'exécution de cette commande, l'enregistrement est le seul de la sélection courante.

Si *enregistrement* est inférieur au plus petit numéro d'enregistrement ou supérieur au plus grand numéro d'enregistrement de la base, 4D génère un message d'erreur indiquant que le numéro est hors intervalle. Si *enregistrement* est égal au numéro d'un enregistrement supprimé, 4D retourne l'erreur -10503 et la sélection courante devient vide.

Exemple

Référez-vous à l'exemple de la commande **Numero enregistrement**.

CREER ENREGISTREMENT

CREER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table dans laquelle créer un enregistrement ou Table par défaut si ce paramètre est omis

Description

CREER ENREGISTREMENT crée un nouvel enregistrement vide pour *laTable*, mais ne l'affiche pas à l'écran. Vous devez utiliser la commande **AJOUTER ENREGISTREMENT** pour créer un nouvel enregistrement et l'afficher dans un formulaire entrée.

Utilisez **CREER ENREGISTREMENT** plutôt que **AJOUTER ENREGISTREMENT** lorsque les valeurs de l'enregistrement sont entrées par programmation. Le nouvel enregistrement devient l'enregistrement courant mais la sélection courante n'est pas modifiée.

L'enregistrement est créé uniquement en mémoire et doit être sauvegardé à l'aide de **STOCKER ENREGISTREMENT**. Si vous changez d'enregistrement courant (par exemple à la suite d'une recherche) avant la sauvegarde, l'enregistrement créé est perdu.

Note : Cette commande ne nécessite pas que *laTable* soit en mode lecture/écriture. Elle peut être utilisée même lorsque la table est en mode lecture seulement (cf. section **Verrouillage d'enregistrements**).

Exemple

L'exemple suivant archive les enregistrements datant de plus de 30 jours. Cette opération est réalisée par la création d'enregistrements dans une table d'archive. Une fois l'opération terminée, les enregistrements archivés sont supprimés de la table *[Comptes]* :

```
  ` Recherche des enregistrements datant de plus de 30 jours
CHERCHER([Comptes];[Comptes]Saisie <(Date du jour 30))
Boucle($vlRecord;1;Enregistrements trouves([Comptes])) ` Boucle une fois par enregistrement
   CREER ENREGISTREMENT([Archives]) ` Création d'un nouvel enregistrement d'archive
   [Archive]Numéro:=[Comptes]Number ` Copie des champs dans l'archive
   [Archive]Saisie :=[Comptes]Saisie
   [Archive]Montant:=[Comptes]Montant
   STOCKER ENREGISTREMENT([Archive]) ` Sauvegarde de l'enregistrement d'archive
   ENREGISTREMENT SUIVANT([Comptes]) ` Passage à l'enregistrement de compte suivant
Fin de boucle
SUPPRIMER SELECTION([Comptes]) ` Suppression des enregistrements
```

DEPILER ENREGISTREMENT

DEPILER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle dépiler l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

DEPILER ENREGISTREMENT charge le premier enregistrement de la pile d'enregistrements de *laTable*, et en fait l'enregistrement courant.

Si vous empilez un enregistrement puis créez une nouvelle sélection courante ne contenant plus l'enregistrement empilé, et enfin dépilez l'enregistrement, vous obtenez la situation dans laquelle l'enregistrement courant ne se trouve pas dans la sélection courante. Si vous souhaitez faire de l'enregistrement empilé la sélection courante, utilisez la commande **ENREGISTREMENT SELECTION**.
Si vous utilisez une routine qui déplace le pointeur d'enregistrement courant avant de sauvegarder l'enregistrement, vous perdrez la copie empilée en mémoire.

Exemple

L'exemple suivant récupère l'enregistrement d'un client dans la pile :

```
DEPILER ENREGISTREMENT([Clients]) ` Dépiler l'enregistrement
```

DUPLIQUER ENREGISTREMENT

DUPLIQUER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de l'enregistrement à dupliquer ou Table par défaut si ce paramètre est omis

Description

DUPLIQUER ENREGISTREMENT duplique l'enregistrement courant de *laTable*. Ce nouvel enregistrement devient l'enregistrement courant. S'il n'y a pas d'enregistrement courant, **DUPLIQUER ENREGISTREMENT** ne fait rien. Appelez la commande **STOCKER ENREGISTREMENT** pour sauvegarder le nouvel enregistrement.

DUPLIQUER ENREGISTREMENT peut être exécuté pendant la saisie des données. Vous pouvez donc dupliquer l'enregistrement qui est affiché à l'écran. N'oubliez pas que vous devez d'abord appeler **STOCKER ENREGISTREMENT** si vous voulez sauvegarder les modifications apportées à l'enregistrement original.

Note de compatibilité : A compter de la version 11 de 4D, cette commande ne prend plus en charge les sous-tables.

EMPILER ENREGISTREMENT

EMPILER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle empiler l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

EMPILER ENREGISTREMENT "empile" une copie de l'enregistrement courant de *laTable* dans la pile d'enregistrements de la table. **EMPILER ENREGISTREMENT** peut être exécuté avant qu'un enregistrement soit sauvegardé.

Si vous empilez un enregistrement non verrouillé, il sera verrouillé pour tous les autres process et utilisateurs jusqu'à ce que vous le "dépiliez" (c'est-à-dire que vous le déchargiez de la pile).

Note de compatibilité : A compter de la version 11 de 4D, cette commande ne prend plus en charge les sous-tables.

Exemple

L'exemple suivant empile l'enregistrement d'un client :

```
EMPILER ENREGISTREMENT([Client]) ` Placer l'enregistrement du client dans la pile
```

Enregistrement charge

Enregistrement charge {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table de l'enregistrement à examiner ou Table par défaut si ce paramètre est omis
Résultat	Booléen	⇐	Vrai si l'enregistrement est chargé, Faux sinon

Description

La commande **Enregistrement charge** retourne Vrai si l'enregistrement courant de *laTable* est chargé dans le process en cours.

4D Server : En principe, lorsque des tables sont liées par des liens automatiques, les enregistrements courants des tables liées sont automatiquement chargés (cf. **Présentation des liens**). Toutefois, pour des raisons d'optimisation, 4D Server ne charge ces enregistrements qu'au moment où c'est nécessaire, par exemple lors de la lecture ou de l'affectation d'un champ de l'enregistrement lié. Par conséquent, dans ce contexte la commande **Enregistrement charge** retournera Faux en mode distant (elle retourne Vrai en mode local).

Exemple

Au lieu d'utiliser les actions automatiques "Enregistrement suivant" ou "Enregistrement précédent", vous voulez écrire dans les méthodes de boutons sans action des instructions spécifiant que le bouton "Suivant" affiche le début de la sélection si la fin de la sélection est atteinte et que le bouton "Précédent" affiche la fin de la sélection si le début est atteint.

```
  ` Méthode objet du bouton sans action "PRÉCÉDENT"
Si(Evenement formulaire=Sur clic)
   ENREGISTREMENT PRECEDENT([Groupe])
   Si(Non(Enregistrement charge([Groupe])))
      ALLER DANS SELECTION([Groupe];Enregistrements trouves([Groupe]))
  `Aller au dernier enregistrement de la sélection
   Fin de si
Fin de si

  ` Méthode objet du bouton sans action "SUIVANT"
Si(Evenement formulaire=Sur clic)
   ENREGISTREMENT SUIVANT([Groupe])
   Si(Non(Enregistrement charge([Groupe])))
      ALLER DANS SELECTION([Groupe];1) `Aller au premier enregistrement de la sélection
   Fin de si
Fin de si
```

Enregistrement modifie

Enregistrement modifie {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	➡	Table de laquelle tester si l'enregistrement courant a été modifié ou Table par défaut si paramètre omis
Résultat	Booléen	⮌	L'enregistrement a été modifié (Vrai) ou L'enregistrement n'a pas été modifié (Faux)

Description

Enregistrement modifie retourne Vrai si l'enregistrement courant de *laTable* a été modifié et non encore stocké. Sinon, elle retourne Faux. Cette fonction vous permet de déterminer rapidement s'il faut stocker l'enregistrement. Dans les formulaires entrée, vous pouvez effectuer le test avant d'aller à l'enregistrement suivant.

A noter que cette fonction retourne toujours Vrai dans les contextes suivants :

- l'enregistrement courant est un nouvel enregistrement,
- après l'exécution des commandes **EMPILER ENREGISTREMENT** et **DEPILER ENREGISTREMENT**,
- dès qu'une valeur a été affectée à un champ de l'enregistrement, même s'il s'agit d'une valeur identique à la précédente. Par exemple, **Enregistrement modifie** retourne Vrai après l'exécution de l'instruction suivante :

```
[Table_1]Champ_1:=[Table_1]Champ_1
```

Exemple

L'exemple suivant montre une utilisation typique de **Enregistrement modifie** :

```
Si(Enregistrement modifie([Clients]))
   STOCKER ENREGISTREMENT([Clients])
Fin de si
```

Enregistrements dans table

Enregistrements dans table {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	→	Table de laquelle retourner le nombre total d'enregistrements ou Table par défaut si ce paramètre est omis
Résultat	Entier long	←	Nombre total d'enregistrements dans table

Description

Enregistrements dans table retourne le nombre total d'enregistrements que contient *laTable*. Par opposition, **Enregistrements trouves** retourne le nombre d'enregistrements de la sélection courante uniquement. Lorsque cette commande est utilisée dans une transaction, les enregistrements éventuellement créés pendant la transaction sont comptabilisés.

Exemple

L'exemple suivant affiche une alerte indiquant le nombre d'enregistrements de la table :

```
ALERTE("Il y a "+Chaine(Enregistrements dans table([Personnes]))+" enregistrements dans la table.")
```

Nouvel enregistrement

Nouvel enregistrement {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table de l'enregistrement à examiner ou Table par défaut si ce paramètre est omis
Résultat	Booléen	⇐	Vrai si l'enregistrement est en cours de création, Faux sinon

Description

La commande **Nouvel enregistrement** retourne Vrai lorsque l'enregistrement courant de *laTable* est en cours de création et n'a pas encore été sauvegardé dans le process courant.

Note de compatibilité : Il est possible d'obtenir la même information avec la commande existante **Numero enregistrement**, en testant si elle retourne -3. Toutefois, il est vivement conseillé d'utiliser dans ce cas **Nouvel enregistrement** plutôt que **Numero enregistrement**. En effet, la commande **Nouvel enregistrement** assure une meilleure compatibilité avec les futures versions de 4D.

4D Server : Cette commande retourne un résultat différent dans le contexte de l'événement formulaire Sur validation suivant qu'elle est exécutée sur 4D en mode local ou 4D en mode distant. En mode local, la commande retourne Faux (l'enregistrement est considéré comme déjà créé). En mode distant, la commande retourne Vrai car dans ce cas, l'enregistrement est également déjà créé sur le serveur mais l'information n'a pas encore été envoyée au client.

Exemple

Les deux instructions suivantes sont identiques, la seconde est conseillée pour que le code reste compatible avec les prochaines versions de 4D :

```
Si(Numero enregistrement([Table])=-3) `Déconseillé
  ` ...
Fin de si

Si(Nouvel enregistrement([Table])) `Conseillé
  ` ...
Fin de si
```

Numero enregistrement

Numero enregistrement {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	➡	Table de laquelle vous souhaitez obtenir le numéro de l'enregistrement courant ou Table par défaut si ce paramètre est omis
Résultat	Entier long	⮌	Numéro d'enregistrement courant

Description

Numero enregistrement retourne le numéro de l'enregistrement courant de *laTable*. S'il n'y a pas d'enregistrement courant, par exemple si le pointeur d'enregistrement se trouve avant ou après la sélection courante, **Numero enregistrement** retourne -1. S'il s'agit d'un nouvel enregistrement qui n'a pas encore été sauvegardé, **Numero enregistrement** retourne -3.

Les numéros d'enregistrements peuvent varier. Par exemple, les numéros des enregistrements supprimés sont réutilisés.

4D Server : Cette commande retourne un résultat différent dans le contexte de l'événement formulaire Sur validation suivant qu'elle est exécutée sur 4D en mode local ou 4D en mode distant. En mode local, la commande retourne un numéro d'enregistrement (l'enregistrement est considéré comme déjà créé). En mode distant, la commande retourne -3 car dans ce cas, l'enregistrement est également déjà créé sur le serveur mais l'information n'a pas encore été envoyée au client.

Note : Il est fortement conseillé d'utiliser la commande **Nouvel enregistrement** pour vérifier si un enregistrement est en cours de création.

Exemple

L'exemple suivant sauvegarde le numéro d'enregistrement courant puis cherche dans la table si un autre enregistrement a la même valeur :

```
$NumEnreg :=Numero enregistrement([Personnes]) ` Obtenir le numéro d'enregistrement
  ` Est-ce que quelqu'un d'autre a le même nom ?
CHERCHER([Personnes];[Personnes]Nom=[Personnes]Nom)
  ` Afficher une alerte avec le nombre de personnes qui ont le même nom
ALERTE("Il existe "+Chaine(Enregistrements trouves([Personnes])+" personnes du même nom.")
ALLER A ENREGISTREMENT([Personnes];$NumEnreg) ` Retourner à l'enregistrement original
```

Numerotation automatique

Numerotation automatique {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	➡	Table à numéroter automatiquement ou Table par défaut si ce paramètre est omis
Résultat	Entier long	⮌	Numéro automatique

Description

Numerotation automatique retourne le prochain numéro automatique de *laTable*. Ce numéro est unique pour chaque table. C'est une valeur qui ne se répète pas et qui est incrémentée à chaque enregistrement nouvellement créé dans la table(*).

(*) Pour des raisons d'optimisation, la numérotation automatique est activée uniquement au premier appel de la commande **Numerotation automatique** ou d'une des fonctions qui y accèdent (cf. ci-dessous). De plus, le compteur peut être réinitialisé via **FIXER PARAMETRE BASE**. Par conséquent, la valeur retournée ne correspond pas nécessairement au nombre d'enregistrements ayant été créés dans *laTable*.

Par défaut, la numérotation commence à 1 ; vous pouvez toutefois modifier la numérotation automatique des enregistrements de *laTable* à l'aide de la commande **FIXER PARAMETRE BASE**.

Note : S'il n'y a pas d'enregistrement courant et que la numérotation a été modifiée via la commande **FIXER PARAMETRE BASE**, le numéro est bien réservé pour la prochaine création d'enregistrement mais ne sera retourné par la fonction **Numerotation automatique** que lorsque la commande **STOCKER ENREGISTREMENT** sera effectivement appelée.

Le numéro retourné par cette fonction pour *laTable* est identique à celui généré si vous avez coché l'option **Incrémentation auto** dans l'Inspecteur de Structure pour un champ de *laTable* ou si vous fixez #N comme valeur par défaut pour un champ de *laTable* dans un formulaire (référez-vous au manuel *Mode Développement* de 4D).

Note : L'incrémentation automatique peut également être définie via l'attribut SQL AUTO_INCREMENT.

La fonction **Numerotation automatique** est utile dans les cas suivants :

- Si la numérotation doit s'incrémenter d'un nombre supérieur à 1
- Si le numéro doit reprendre une partie d'un code

Pour stocker ce numéro à l'aide d'une méthode, il faut créer un champ de type Entier long dans la table et y affecter la numérotation automatique.
Si la numérotation doit débuter à une valeur différente de 1, ajoutez simplement la différence à la fonction **Numerotation automatique**. Par exemple, si le numéro doit commencer à 1000, vous pouvez utiliser la ligne de code suivante pour affecter le numéro :

```
[Table1]NumAuto:=Numerotation automatique([Table1])+999
```

Exemple

L'exemple suivant fait partie d'une méthode formulaire. Ces lignes de code testent s'il s'agit d'un nouvel enregistrement (si le numéro de facture est égal à une chaîne vide). Si l'enregistrement est nouveau, un numéro est affecté à la facture. Ce numéro de facture est construit avec deux informations : le numéro unique et le numéro de référence de l'utilisateur, qui était saisi à l'ouverture de la base. Le numéro unique est formaté en tant que chaîne avec une longueur de cinq caractères :

```
Si([Factures]NumFacture="") ` S'il s'agit d'une nouvelle facture, créer un numéro de facture
  ` Le numéro de facture est une chaîne qui se termine par le numéro de référence de l'utilisateur
   [Factures]NumFacture:=Chaine(Numerotation automatique;"00000")+[Factures]Utilisateur
Fin de si
```

STOCKER ENREGISTREMENT

STOCKER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de l'enregistrement à stocker ou Table par défaut si ce paramètre est omis

Description

STOCKER ENREGISTREMENT sauvegarde l'enregistrement courant de *laTable* pour le process courant. S'il n'y a pas d'enregistrement courant, la commande est ignorée.

Vous pouvez utiliser **STOCKER ENREGISTREMENT** pour sauvegarder un enregistrement créé ou modifié par programmation. Lorsqu'un enregistrement a été modifié puis validé par un utilisateur dans un formulaire, il n'est pas nécessaire de le sauvegarder à l'aide de **STOCKER ENREGISTREMENT**. En revanche, un enregistrement modifié puis annulé par l'utilisateur peut malgré tout être sauvegardé avec **STOCKER ENREGISTREMENT**.

Si vous appelez la commande **STOCKER ENREGISTREMENT** alors qu'aucun champ n'a été modifié dans l'enregistrement, la commande ne fait rien (le trigger n'est pas appelé).

L'utilisation de **STOCKER ENREGISTREMENT** est nécessaire dans les cas suivants :

- Pour sauvegarder un enregistrement créé par les commandes **CREER ENREGISTREMENT** ou **DUPLIQUER ENREGISTREMENT**,
- Pour sauvegarder des données issues de la commande **RECEVOIR ENREGISTREMENT**,
- Pour sauvegarder un enregistrement modifié par une méthode,
- Pour sauvegarder un enregistrement contenant un sous-enregistrement ayant été créé ou modifié par la commande **_o_AJOUTER SOUS ENREGISTREMENT**, **_o_CRÉER SOUS ENREGISTREMENT**, ou **_o_MODIFIER SOUS ENREGISTREMENT**,
- Pendant la saisie de données, pour sauvegarder l'enregistrement affiché avant d'appeler une commande qui change l'enregistrement courant,
- Pendant la saisie de données, pour sauvegarder l'enregistrement courant.

Vous ne devez pas appeler **STOCKER ENREGISTREMENT** dans l'événement formulaire Sur validation d'un enregistrement qui a été validé, sinon l'enregistrement est sauvegardé deux fois.

Exemple

L'exemple suivant est une partie d'une méthode récupérant des enregistrements d'un fichier. Dans cette partie, les enregistrements sont reçus puis, si l'opération s'est correctement déroulée, sauvegardés :

```
RECEVOIR ENREGISTREMENT([Clients]) ` Réception de l'enregistrement à partir du disque
Si(OK=1) ` Si l'enregistrement a été correctement reçu…
   STOCKER ENREGISTREMENT([Clients]) ` Le sauvegarder
Fin de si
```

SUPPRIMER ENREGISTREMENT

SUPPRIMER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle supprimer l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

SUPPRIMER ENREGISTREMENT supprime de *laTable* l'enregistrement courant du process en cours. S'il n'y a pas d'enregistrement courant pour *laTable* dans le process, **SUPPRIMER ENREGISTREMENT** ne fait rien. Dans un formulaire, vous pouvez créer un bouton 'Supprimer enregistrement' et lui assigner l'action automatique correspondante, plutôt que d'utiliser cette commande.

Notes :

- Si l'enregistrement courant est déchargé de la mémoire avant l'appel à **SUPPRIMER ENREGISTREMENT** (par exemple suite à un **LIBERER ENREGISTREMENT**), la sélection courante de *laTable* est vide à l'issue de la suppression.
- La commande **SUPPRIMER ENREGISTREMENT** ne fait rien si la table est en mode **LECTURE SEULEMENT**, indépendamment de l'état verrouillé ou non de l'enregistrement à supprimer.

La suppression d'enregistrements est une opération définitive et ne peut être annulée.

Lorsqu'un enregistrement est supprimé, son numéro interne est réutilisé lors de la création de nouveaux enregistrements. Par conséquent, n'utilisez pas ces numéros comme identifiants de vos enregistrements si votre base permet la suppression d'enregistrements.

Exemple

L'exemple suivant permet de supprimer l'enregistrement d'un employé. La méthode demande à l'utilisateur le numéro de l'employé à supprimer, recherche l'enregistrement correspondant puis le supprime :

```
vCherch:=Demander("Numéro de l'employé à supprimer :") //On récupère un numéro d'identification
Si(OK=1)
   CHERCHER([Employés];[Employés]Numéro=vCherch) //Trouver l'employé
   SUPPRIMER ENREGISTREMENT([Employés]) //Suppression de l'enregistrement
Fin de si
```

Enregistrements (verrouillage)

- Verrouillage d'enregistrements
- CHARGER ENREGISTREMENT
- Enregistrement verrouille
- Etat lecture seulement
- LECTURE ECRITURE
- LECTURE SEULEMENT
- LIBERER ENREGISTREMENT
- Lire infos enregistrements verrouillés
- VERROUILLE PAR

Verrouillage d'enregistrements

4D et 4D Server gèrent automatiquement les bases en empêchant les conflits entre les process ou entre les utilisateurs. Deux utilisateurs ou deux process ne peuvent pas modifier en même temps le même enregistrement ou le même objet. Le second utilisateur ou process peut toutefois accéder simultanément en "lecture seulement" à l'enregistrement ou à l'objet.

Vous devez utiliser les commandes multi-utilisateurs de cette section dans plusieurs cas :

- Modification d'enregistrements par programmation,
- Utilisation d'une interface utilisateur personnalisée pour les opérations multi-utilisateurs,
- Sauvegarde de modifications reliées entre elles dans une transaction.

Il y a trois concepts importants à maîtriser lorsqu'on utilise des commandes dans une base multi-process :

1. Dans un process, chaque table est soit en mode "lecture seulement" soit en mode "lecture/écriture".
2. Les enregistrements sont verrouillés quand ils sont chargés et déverrouillés quand ils sont libérés.
3. Un enregistrement verrouillé ne peut être modifié.

Dans les paragraphes suivants, la personne effectuant une opération dans la base multi-utilisateurs est l'utilisateur local. Les autres personnes utilisant la base sont appelées les autres utilisateurs. La discussion est menée du point de vue de l'utilisateur local. De même, du point de vue du multi-process, le process exécutant une opération dans la base est le process courant. Tout autre process en cours d'exécution est désigné comme un autre process. La discussion est menée du point de vue du process courant.

Enregistrements verrouillés

Un enregistrement verrouillé ne peut pas être modifié par l'utilisateur local ou le process courant. Un enregistrement verrouillé peut être chargé, mais pas modifié. Un enregistrement est verrouillé quand l'un des autres utilisateurs ou process a chargé l'enregistrement pour effectuer une modification ou quand l'enregistrement est empilé. Seul l'utilisateur qui modifie l'enregistrement perçoit l'enregistrement comme étant non verrouillé. Tous les autres utilisateurs et process perçoivent l'enregistrement comme étant verrouillé, et donc indisponible pour modification. Une table doit être en mode lecture/écriture pour qu'un enregistrement puisse être chargé déverrouillé (modifiable).

Etats Lecture seulement et Lecture/écriture

Chaque table d'une base est soit en mode lecture/écriture, soit en mode lecture seulement pour chaque utilisateur et process de la base. **Lecture seulement** signifie que les enregistrements de la table peuvent être chargés mais non modifiés. **Lecture/écriture** signifie que les enregistrements de la table peuvent être chargés et modifiés par un utilisateur/process, si aucun autre utilisateur/process n'a prélablement verrouillé l'enregistrement.

Notez que si vous changez l'état d'une table, celui-ci prend effet pour le prochain enregistrement chargé. Si un enregistrement est déjà chargé, la modification de l'état de la table ne l'affecte pas.

Etat lecture seulement

Lorsqu'une table est en lecture seulement et qu'un enregistrement est chargé, l'enregistrement est toujours verrouillé. En d'autres termes, l'enregistrement peut être affiché, imprimé et utilisé de diverses façons, mais ne peut être modifié.

Notez que le mode lecture seulement ne s'applique qu'à la modification d'enregistrements existants. Le mode lecture seulement n'affecte pas la création de nouveaux enregistrements : vous pouvez toujours ajouter des enregistrements à une table en lecture seulement en utilisant les commandes **CREER ENREGISTREMENT** et **AJOUTER ENREGISTREMENT** ou les commandes de menu du mode Développement (dans ce cas les enregistrements en cours de création sont verrouillés pour les autres process/utilisateurs). A noter que la commande **TABLEAU VERS SELECTION** n'est pas affectée par le mode lecture seulement car elle permet aussi bien de créer que de modifier des enregistrements.

4D met automatiquement une table en mode lecture seulement pour les commandes qui ne requièrent pas d'accès en écriture aux enregistrements. Ces commandes sont **VISUALISER SELECTION**, **VALEURS DISTINCTES**, **ECRITURE DIF**, **ECRITURE SYLK**, **EXPORTER TEXTE**, **_o_GRAPHE SUR SELECTION**, **IMPRIMER SELECTION**, **IMPRIMER ETIQUETTES**, **QR ETAT**, **SELECTION VERS TABLEAU**, **SELECTION LIMITEE VERS TABLEAU**.

Vous pouvez connaître à tout moment l'état d'une table à l'aide de la fonction **Etat lecture seulement**.

Avant d'exécuter ces commandes, 4D sauvegarde l'état courant de la table (lecture seulement ou lecture/écriture) dans le process courant. Une fois la commande exécutée, l'état initial est rétabli.

Etat lecture/écriture

Lorsqu'une table est en lecture/écriture et qu'un enregistrement est chargé, l'enregistrement sera non verrouillé si aucun autre utilisateur ne l'a préalablement chargé. Si l'enregistrement est verrouillé par un autre utilisateur, l'enregistrement est chargé verrouillé et ne peut être modifié par l'utilisateur local.

Une table doit être en mode lecture/écriture et l'enregistrement doit être chargé pour qu'il soit déverrouillé et donc modifiable.

Si un utilisateur charge un enregistrement d'une table en mode lecture/écriture, aucun autre utilisateur ne peut charger cet enregistrement pour modification. Les autres utilisateurs peuvent toujours, cependant, ajouter des enregistrements dans la table, soit manuellement en mode Développement, soit par les commandes **CREER ENREGISTREMENT** ou **AJOUTER ENREGISTREMENT**.

Le mode lecture/écriture est le mode par défaut pour toutes les tables quand une base est ouverte et un nouveau process démarré.

Changer l'état d'une table

Vous pouvez utiliser les commandes **LECTURE SEULEMENT** et **LECTURE ECRITURE** pour changer l'état d'une table. Si vous voulez changer l'état d'une table pour mettre un enregistrement en lecture seulement ou lecture/écriture, vous devez exécuter la commande avant le chargement de l'enregistrement. Un enregistrement déjà chargé n'est pas affecté par les commandes **LECTURE SEULEMENT** et **LECTURE ECRITURE**.

Chaque process dispose de son propre état (lecture seulement ou lecture/écriture) pour chaque table dans la base.

Par défaut, si vous n'utilisez pas la commande **LECTURE SEULEMENT**, toutes les tables sont en mode lecture/écriture.

Charger, modifier et libérer des enregistrements

Pour que l'utilisateur local puisse modifier un enregistrement, la table doit être en mode lecture/écriture et l'enregistrement doit être chargé et non verrouillé.

Chacune des commandes chargeant un enregistrement courant (s'il y en a un) — telles que **ENREGISTREMENT SUIVANT**, **CHERCHER**, **TRIER**, **CHARGER SUR LIEN**, etc — définit l'état verrouillé ou non verrouillé de l'enregistrement. L'enregistrement est chargé en fonction de l'état courant de sa table (lecture seulement ou lecture/écriture) et sa propre disponibilité. Un enregistrement peut aussi être chargé d'une table liée par une commande activant le lien automatique.

Lorsqu'une table est en mode lecture seulement pour un process ou un utilisateur, tout enregistrement de cette table est chargé en mode lecture seulement, ce qui signifie qu'il ne pourra pas être modifié ou supprimé par ce process ou utilisateur. Ce mode est recommandé pour la visualisation et la recherche de données car il n'empêche pas les autres process ou utilisateurs d'accéder en mode lecture/écriture aux enregistrements de la table si nécessaire.

Lorsqu'une table est en mode lecture/écriture pour un process ou un utilisateur, tout enregistrement de cette table est chargé en mode lecture/écriture à condition qu'aucun autre process ou utilisateur ne l'ait préalablement verrouillé. Si l'enregistrement a pu être chargé en lecture/écriture, il est déverrouillé pour le process ou utilisateur courant (il peut être modifié et sauvegardé) et il est verrouillé pour les autres process ou utilisateurs. Une table doit être placée en mode lecture/écriture par le process/utilisateur avant tout chargement d'un enregistrement pour modification et sauvegarde.

Si un enregistrement doit être modifié, utilisez la fonction **Enregistrement verrouille** pour tester s'il est ou non verrouillé par un autre utilisateur. Si l'enregistrement est verrouillé (**Enregistrement verrouille** retourne Vrai), chargez l'enregistrement avec la commande **CHARGER ENREGISTREMENT** et testez de nouveau le verrouillage. Répétez l'opération jusqu'à ce que l'enregistrement soit libéré (**Enregistrement verrouille** retourne Faux).

Lorsque vous en avez terminé avec les modifications à apporter à un enregistrement, il doit être libéré pour les autres utilisateurs (et donc déverrouillé) avec la commande **LIBERER ENREGISTREMENT**. Si un enregistrement n'est pas libéré, il reste verrouillé pour tous les autres utilisateurs jusqu'à ce qu'un nouvel enregistrement courant soit sélectionné. Changer l'enregistrement courant d'une table libère automatiquement l'enregistrement courant précédent. Vous devez explicitement appeler **LIBERER ENREGISTREMENT** si vous ne changez pas l'enregistrement courant. Ce principe ne s'applique qu'aux enregistrements existants : quand un enregistrement est créé, il peut être sauvegardé quel que soit l'état de la table auquel il appartient.

Note : Lorsqu'elle est utilisée dans une transaction, la commande **LIBERER ENREGISTREMENT** libère l'enregistrement courant pour le process qui gère la transaction uniquement. Pour les autres process, l'enregistrement reste verrouillé tant que la transaction n'est pas validée (ou annulée).

Utilisez la commande **VERROUILLE PAR** pour savoir quel utilisateur ou process a verrouillé un enregistrement.

Note : Une bonne pratique consiste à placer toutes les tables en mode lecture seulement au démarrage de chaque process (via la syntaxe **LECTURE SEULEMENT(*)**) puis à placer chaque table en mode lecture/écriture uniquement en cas de besoin. L'accès aux tables en mode lecture seulement est plus rapide et plus économe en mémoire. De plus, le changement d'état d'une table est optimisé en mode client/serveur car il ne provoque pas de trafic réseau supplémentaire : l'information est envoyée au serveur uniquement lors de l'exécution d'une commande nécessitant l'accès adéquat à la table.

Boucles pour charger des enregistrements non verrouillés

Cet exemple présente la boucle la plus simple pour charger un enregistrement non verrouillé :

```
LECTURE ECRITURE([Clients]) ` Placer la table en lecture/écriture
Repeter ` Boucler jusqu'à ce que l'enregistrement soit déverrouillé
   CHARGER ENREGISTREMENT([Clients]) ` Charger et verrouiller l'enregistrement
Jusque(Non(Enregistrement verrouille([Clients])))
```

La boucle s'exécute jusqu'à ce que l'enregistrement soit libéré.

Une boucle de ce type ne s'emploie que lorsqu'il est peu probable que l'enregistrement soit verrouillé par quelqu'un d'autre, puisque l'utilisateur aurait à attendre la fin de la boucle. Aussi, une telle boucle est rarement utilisée, sauf si l'enregistrement n'est modifiable que par méthode.

Cet exemple utilise la boucle précédente pour charger un enregistrement non verrouillé et modifier l'enregistrement :

```
LECTURE ECRITURE([Stocks])
Repeter ` Boucle jusqu'à ce que l'enregistrement soit déverrouillé
   CHARGER ENREGISTREMENT([Stocks]) ` Charger l'enregistrement et le verrouiller
Jusque(Non(Enregistrement verrouille([Stocks])))
[Stocks]Part Qté :=[Stocks]Part Qté 1  ` Modifier l'enregistrement
STOCKER ENREGISTREMENT([Stocks]) ` Sauvegarder l'enregistrement
LIBERER ENREGISTREMENT([Stocks]) ` Laisser d'autres utilisateurs le modifier
LECTURE SEULEMENT([Stocks])
```

La commande **MODIFIER ENREGISTREMENT** prévient automatiquement l'utilisateur si un enregistrement est verrouillé, et interdit sa modification. L'exemple suivant évite la notification automatique par un test préalable de l'enregistrement avec la fonction **Enregistrement verrouille**. Si l'enregistrement est verrouillé, l'utilisateur peut annuler.

Cet exemple teste si l'enregistrement courant est verrouillé pour la table [Commandes]. Si c'est le cas, le process est endormi par la méthode pendant quelques instants. Cette technique peut être utlisée à la fois dans un développement multi-utilisateurs et multi-process :

```
Repeter
   LECTURE SEULEMENT([Commandes]) ` Vous n'avez pas besoin de lecture/écriture pour le moment
   CHERCHER([Commandes])
 ` Si la recherche est terminée et que des enregistrements sont retournés
   Si((OK=1) & (Enregistrements trouves([Commandes])>0))
      LECTURE ECRITURE([Commandes]) ` Mettre la table en mode lecture/écriture
      CHARGER ENREGISTREMENT([Commandes])
      Tant que(Enregistrement verrouille([Commandes]) & (OK=1))
 ` Si l'enregistrement est verrouillé,
 ` boucler jusqu'à ce que l'enregistrement soit libéré
 ` Par qui l'enregistrement est-il verrouillé ?
         VERROUILLE PAR([Commandes];$Process;$Utilisateur;$SessionUtil;$Nom)
         Si($Process=-1) ` L'enregistrement a-t-il été détruit?
            ALERTE("L'enregistrement a été détruit dans l'intervalle.")
            OK:=0
         Sinon
            Si($Utilisateur="") ` Etes en mode simple utilisateur ?
               $Utilisateur:="vous-même"
            Fin de si
            CONFIRMER("L'enregistrement est utilisé par "+$Utilisateur+" dans le "+$Nom+" Process.")
            Si(OK=1) ` Si vous voulez attendre quelques secondes
```

```
                ENDORMIR PROCESS(Numero du process courant;120) `Attente
                CHARGER ENREGISTREMENT([Commandes]) ` Essayez de charger l'enregistrement
             Fin de si
          Fin de si
       Fin tant que
       Si(OK=1) ` L'enregistrement est libéré
          MODIFIER ENREGISTREMENT([Commandes]) ` Vous pouvez modifier l'enregistrement
          LIBERER ENREGISTREMENT([Commandes])
       Fin de si
       LECTURE SEULEMENT([Commandes]) ` Retour à lecture seulement
       OK:=1
    Fin de si
 Jusque(OK=0)
```

Comportement des commandes en cas d'enregistrement verrouillé

Certaines commandes du langage effectuent des actions particulières lorsqu'elles rencontrent un enregistrement verrouillé. Voici la liste de ces commandes :

- **MODIFIER ENREGISTREMENT** : Cette commande affiche une boîte de dialogue indiquant que l'enregistrement est utilisé. L'enregistrement n'est pas affiché et donc l'utilisateur ne peut le modifier. En mode Développement, l'enregistrement est visible en lecture seulement.
- **MODIFIER SELECTION** : Cette commande se comporte normalement à ceci près que lorsque l'utilisateur double-clique sur un enregistrement pour le modifier, **MODIFIER SELECTION** affiche une boîte de dialogue indiquant que l'enregistrement est utilisé et n'autorise que sa lecture.
- **APPLIQUER A SELECTION** : Cette commande charge un enregistrement verrouillé, mais ne le modifie pas. **APPLIQUER A SELECTION** peut être utilisée pour lire les informations de la table sans problème. Si la commande rencontre un enregistrement verrouillé, celui-ci est placé dans l'ensemble système **LockedSet**.
- **SUPPRIMER SELECTION** : Cette commande ne supprime pas l'enregistrement verrouillé, elle l'ignore simplement. L'enregistrement verrouillé est placé dans l'ensemble **LockedSet**.
- **SUPPRIMER ENREGISTREMENT** : Cette commande est ignorée si l'enregistrement est verrouillé. Aucune erreur n'est retournée. Vous devez vous assurer que l'enregistrement est non verrouillé avant d'exécuter cette commande.
- **STOCKER ENREGISTREMENT** : Cette commande est ignorée si l'enregistrement est verrouillé. Aucune erreur n'est retournée. Vous devez vous assurer que l'enregistrement est non verrouillé avant d'exécuter cette commande.
- **TABLEAU VERS SELECTION** : Cette commande ne sauvegarde pas les enregistrements verrouillés. Si elle rencontre un enregistrement verrouillé, celui-ci est placé dans l'ensemble **LockedSet**.
- **ALLER A ENREGISTREMENT** : Dans une base multi-utilisateurs/multi-process, des enregistrements peuvent être ajoutés ou supprimés par d'autres utilisateurs/process. Les numéros d'enregistrement peuvent donc varier. Soyez prudent lorsque vous référencez un enregistrement par son numéro dans une base multi-utilisateurs.
- **Présentation des ensembles** : Soyez prudent lors de la manipulation d'ensembles puisque les informations sur lesquelles était basée la construction de l'ensemble peuvent avoir été modifiées par un autre utilisateur ou process.

CHARGER ENREGISTREMENT

CHARGER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle charger l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

CHARGER ENREGISTREMENT charge l'enregistrement courant de *laTable*. S'il n'y a pas d'enregistrement courant, **CHARGER ENREGISTREMENT** ne fait rien.

Il est alors utile d'appeler la fonction **Enregistrement verrouille** pour déterminer si l'enregistrement peut être modifié :

- Si la table est en mode Lecture seulement, la fonction retourne Vrai et vous ne pouvez pas modifier l'enregistrement.
- Si la table est en mode Lecture/écriture mais si l'enregistrement est déjà verrouillé, il sera en mode Lecture seulement et vous ne pouvez pas le modifier.
- Si la table est en mode Lecture/écriture et si l'enregistrement n'est pas verrouillé, vous pouvez le modifier dans le process courant. La fonction **Enregistrement verrouille** retournera Vrai pour tous les autres utilisateurs et process.

Note : Si la commande **CHARGER ENREGISTREMENT** est exécutée après un **LECTURE SEULEMENT**, l'enregistrement est automatiquement libéré et chargé, sans qu'il soit nécessaire d'appeler la commande **LIBERER ENREGISTREMENT**.

Vous n'aurez normalement pas besoin d'appeler la commande **CHARGER ENREGISTREMENT**, car toutes les commandes telles que **CHERCHER**, **ENREGISTREMENT SUIVANT**, **ENREGISTREMENT PRECEDENT**, etc., chargent automatiquement l'enregistrement courant.

En environnements multi-utilisateurs et multi-process, lorsque vous devez modifier un enregistrement existant, il vous faut accéder en Lecture/écriture à la table à laquelle appartient l'enregistrement. Lorsqu'un enregistrement verrouillé ne peut être chargé, **CHARGER ENREGISTREMENT** vous permet de tenter à nouveau plus tard de charger l'enregistrement. En utilisant **CHARGER ENREGISTREMENT** dans une boucle, vous pouvez attendre que l'enregistrement devienne accessible en Lecture/écriture.

Astuce : La commande **CHARGER ENREGISTREMENT** peut être utilisée pour recharger l'enregistrement courant dans le contexte d'un formulaire entrée. Toutes les données modifiées sont alors remplacées par leurs valeurs précédentes. Dans ce cas, la commande **CHARGER ENREGISTREMENT** effectue en quelque sorte une annulation globale de la saisie.

Enregistrement verrouille

Enregistrement verrouille {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table de l'enregistrement dont vous voulez tester le verrouillage ou Table par défaut si ce paramètre est omis
Résultat	Booléen	⇐	L'enregistrement est verrouillé (Vrai) ou L'enregistrement n'est pas verrouillé (Faux)

Description

Enregistrement verrouille teste si l'enregistrement courant de *laTable* est verrouillé. Cette fonction vous permet de savoir si un enregistrement est verrouillé ou non, et donc de réagir de manière appropriée, par exemple en laissant à l'utilisateur le choix d'attendre que l'enregistrement soit libéré ou d'annuler l'opération.

Si **Enregistrement verrouille** retourne Vrai, l'enregistrement ne peut être sauvegardé car il est verrouillé par un autre utilisateur, un autre process ou est empilé dans le process courant. La commande **VERROUILLE PAR** indique l'utilisateur ou le numéro du process à l'origine du verrouillage. Dans ce cas, vous devez appeler la commande **CHARGER ENREGISTREMENT** pour tenter à nouveau de charger l'enregistrement, jusqu'à ce que **Enregistrement verrouille** retourne Faux.

Si **Enregistrement verrouille** retourne Faux, l'enregistrement n'est pas verrouillé, ce qui signifie qu'il est verrouillé pour tous les autres utilisateurs. Seul l'utilisateur ayant chargé l'enregistrement ou le process courant peut modifier et sauvegarder l'enregistrement. Une table doit être en mode lecture/écriture si vous voulez modifier les enregistrements qu'elle contient.

Lorsque vous tentez de charger un enregistrement qui a été supprimé, **Enregistrement verrouille** continue de retourner Vrai. Pour éviter d'attendre un enregistrement qui n'existe plus, appelez la commande **VERROUILLE PAR**. Cette commande retourne -1 dans le paramètre *process* si l'enregistrement a été supprimé.

Note : Enregistrement verrouille retourne Faux lorsqu'il n'y a pas d'enregistrement courant dans *table*, c'est-à-dire lorsque **Numero enregistrement** retourne -1.

Au cours d'une transaction, **CHARGER ENREGISTREMENT** et **Enregistrement verrouille** sont souvent appelées pour tester la disponibilité des enregistrements. Si un enregistrement est verrouillé, il suffit d'annuler la transaction.

Etat lecture seulement

Etat lecture seulement {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table pour laquelle il faut tester l'état ou Table par défaut si ce paramètre est omis
Résultat	Booléen	⇐	Accès à la table est lecture seulement (Vrai) ou Accès à la table est lecture-écriture (Faux)

Description

La fonction **Etat lecture seulement** est utilisé pour tester si *laTable* est en mode lecture seulement dans le process où la fonction est appelée. **Etat lecture seulement** retourne Vrai si *laTable* est en lecture seulement, et Faux si *laTable* est en lecture-écriture.

Exemple

L'exemple suivant teste le statut de la table [Factures]. Si elle est en lecture seulement, le mode lecture-écriture lui est appliqué et l'enregistrement courant est rechargé.

```
Si(Etat lecture seulement([Factures]))
   LECTURE ECRITURE([Factures])
   CHARGER ENREGISTREMENT([Factures])
Fin de si
```

Note : L'enregistrement courant est rechargé pour permettre à l'utilisateur de le modifier. En effet, un enregistrement précédemment chargé en mode lecture seulement reste verrouillé jusqu'à ce qu'il soit rechargé en mode lecture-écriture.

LECTURE ECRITURE

LECTURE ECRITURE {(laTable | *)}

Paramètre	Type		Description
laTable | *	Table, Opérateur	⇒	Table à définir en mode lecture/écriture ou * pour toutes les tables ou Table par défaut si ce paramètre est omis

Description

LECTURE ECRITURE place *laTable* en mode lecture/écriture pour le process dans lequel la commande a été appelée. Si vous passez le paramètre optionnel *, toutes les tables sont placées en mode lecture/écriture.

Après un appel à **LECTURE ECRITURE**, lorsqu'un enregistrement est chargé, il n'est pas verrouillé — sauf si un autre utilisateur l'a déjà chargé. Cette commande ne modifie pas le statut des enregistrements déjà chargés, seuls les enregistrements chargés par la suite sont affectés.

Par défaut, toutes les tables sont en mode lecture/écriture.

Utilisez **LECTURE ECRITURE** lorsque vous devez modifier un enregistrement et sauvegarder les modifications. Vous pouvez également appeler cette commande lorsque vous voulez qu'un enregistrement soit verrouillé pour les autres utilisateurs, même si vous ne souhaitez pas effectuer de modifications. Placer une table en mode lecture/écriture vous permet d'empêcher les autres utilisateurs d'effectuer des modifications sur cette table. Cependant, ils peuvent continuer à créer des nouveaux enregistrements.

Note : Cette commande n'est pas rétroactive. Les privilèges de lecture/écriture pour un enregistrement sont définis par ceux de la table au moment où l'enregistrement est chargé. Pour qu'un enregistrement soit chargé en mode lecture/écriture alors que la table est en mode lecture seulement, vous devez placer la table en mode lecture/écriture avant que l'enregistrement soit chargé.

LECTURE SEULEMENT

LECTURE SEULEMENT {(laTable | *)}

Paramètre	Type		Description
laTable | *	Table, Opérateur	⇨	Table à définir en mode lecture seulement ou * pour toutes les tables ou Table par défaut si ce paramètre est omis

Description

LECTURE SEULEMENT place *laTable* en mode lecture seulement pour le process dans lequel la commande a été appelée. Tous les enregistrements chargés par la suite sont verrouillés, aucune modification ne peut leur être apportée. Si vous passez le paramètre optionnel *, toutes les tables sont placées en mode lecture seulement.

Vous pouvez utiliser **LECTURE SEULEMENT** lorsqu'il n'est pas utile de modifier les enregistrements.

Note : Cette commande n'est pas rétroactive. Les privilèges de lecture/écriture pour un enregistrement sont définis par ceux de la table au moment où l'enregistrement est chargé. Pour qu'un enregistrement soit chargé en mode lecture seulement alors que la table est en mode lecture/écriture, vous devez placer la table en mode lecture seulement avant que l'enregistrement soit chargé.

LIBERER ENREGISTREMENT

LIBERER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table pour laquelle l'enregistrement est à libérer ou Table par défaut si ce paramètre est omis

Description

LIBERER ENREGISTREMENT place l'enregistrement courant de *laTable* dans l'état non verrouillé.

Si l'enregistrement n'est pas verrouillé par un utilisateur (mais est verrouillé pour les utilisateurs), **LIBERER ENREGISTREMENT** libère l'enregistrement pour tous les utilisateurs.

Même si **LIBERER ENREGISTREMENT** libère l'enregistrement de la mémoire, celui-ci reste l'enregistrement courant. Lorsqu'un enregistrement devient l'enregistrement courant, le précédent est automatiquement libéré et n'est donc plus verrouillé pour les autres utilisateurs. Vous devez appeler cette commande lorsque vous avez fini de modifier un enregistrement, que vous voulez qu'il reste l'enregistrement courant et qu'il soit accessible aux autres utilisateurs.

Si les enregistrements contiennent une quantité importante de données, de champs Image ou de documents externes (tels que des documents 4D Write ou 4D Draw), il est préférable de ne pas stocker l'enregistrement courant en mémoire sauf si vous avez besoin de le modifier. Dans ce cas, il faut utiliser la commande **LIBERER ENREGISTREMENT**. De cette manière, vous pouvez conserver l'enregistrement courant sans qu'il soit en mémoire. Vous libérez ainsi la mémoire occupée par l'enregistrement, mais vous n'avez pas accès aux valeurs stockées dans les champs. Si vous avez besoin d'exploiter ces valeurs, il faut utiliser la commande **CHARGER ENREGISTREMENT**.

Note : Lorsqu'elle est utilisée dans une transaction, la commande **LIBERER ENREGISTREMENT** libère l'enregistrement courant pour le process qui gère la transaction uniquement. Pour les autres process, l'enregistrement reste verrouillé tant que la transaction n'est pas validée (ou annulée).

Lire infos enregistrements verrouillés

Lire infos enregistrements verrouillés (laTable) -> Résultat

Paramètre	Type		Description
laTable	Table	➡	Table de laquelle vous souhaitez connaître les enregistrements verrouillés
Résultat	Objet	⮌	Description des enregistrements verrouillés (le cas échéant)

Description

La commande **Lire infos enregistrements verrouillés** retourne un *objet* décrivant le ou les enregistrement(s) actuellement verrouillé(s) dans *laTable*.
L'objet retourné contient une propriété "records" qui est un tableau d'objets :

```
{
    "records": [
        objet description,
        (…)
    ]
}
```

Chaque élément de tableau "objet description" identifie un enregistrement verrouillé dans la table spécifiée et contient les propriétés suivantes :

Propriété	Type	Description
contextID	UUID (Chaîne)	UUID du contexte de la base à l'origine du verrouillage
contextAttributes	Objet	Objet contenant des informations semblables à la commande **VERROUILLE PAR** mais appliquées à l'enregistrement, à la différence près que **Lire infos enregistrements verrouillés** retourne uniquement le nom de l'utilisateur défini dans le système (et pas celui de l'utilisateur 4D) ainsi que des informations supplémentaires (voir ci-dessous).
recordNumber	Entier long	Numéro de l'enregistrement verrouillé

L'objet *contextAttributes* est constitué des propriétés suivantes :

Propriété	Type	Description
task_id	Numérique	Numéro de référence du process
user_name	Chaîne	Nom de l'utilisateur défini dans le système d'exploitation
user4d_id	Numérique	Numéro de l'utilisateur 4D(*)
host_name	Chaîne	Nom de la machine hôte
task_name	Chaîne	Nom du process
client_version	Numérique	version de l'application cliente
Uniquement lorsque la commande est exécutée sur 4D Server et si le verrouillage de l'enregistrement provient d'un 4D distant :		
is_remote_context	Booléen	Indique si l'origine du verrouillage est un 4D distant (toujours *true* car non présent dans les autres cas)
client_uid	UUID (Chaîne)	Identifiant UUID du 4D distant à l'origine du verrouillage

(*) Vous pouvez obtenir le nom d'utilisateur 4D à partir de la valeur de *user4d_id* en utilisant le code suivant :

```
LIRE LISTE UTILISATEURS($tabNoms;$tabIDs)
$nom4DUser:=Chercher dans tableau($tabIDs;user4d_id)
```

Note : La commande fonctionne uniquement avec 4D et 4D Server. Elle retourne toujours un objet invalide lorsqu'elle est appelée depuis un 4D distant. Elle peut toutefois être appelée depuis un 4D distant si la méthode d'appel dispose de l'option "Exécuter sur serveur" ; elle retourne dans ce cas les informations relatives au serveur. Lorsqu'elle est appelée depuis un composant, elle s'applique à la base hôte.

Exemple

Vous exécutez le code suivant :

```
$vOlocked :=Lire infos enregistrements verrouillés([Table])
```

Si deux enregistrements sont verrouillés dans la table [Table], l'objet suivant est retourné dans $vOlocked :

```
{
    "records": [
        {
            "contextID": "A9BB84C0E57349E089FA44E04C0F2F25",
            "contextAttributes": {
                "task_id": 8,
                "user_name": "roland",
                "user4d_id": 1,
                "host_name": "iMac de roland",
                "task_name": "P_RandomLock",
                "client_version": -1342106592
            },
            "recordNumber": 1
```

```
        },
        {
            "contextID": "8916338D1B8A4D86B857D92F593CCAC3",
            "contextAttributes": {
                "task_id": 9,
                "user_name": "roland",
                "user4d_id": 1,
                "host_name": "iMac de roland",
                "task_name": "P_RandomLock",
                "client_version": -1342106592
            },
            "recordNumber": 2
        }
    ]
}
```

Si le code est exécuté sur 4D Server et que le verrouillage est causé par un poste client distant, l'objet suivant est retourné dans $vOlocked :

```
{
    "records": [
        {
            "contextID": "B0EC087DC2FA704496C0EA15DC011D1C",
            "contextAttributes": {
                "task_id": 2,
                "user_name": "achim",
                "user4d_id": 1,
                "host_name": "achim-pcwin",
                "task_name": "P_RandomLock",
                "is_remote_context": true,
                "client_uid": "0696E66F6CD731468E6XXX581A87554A",
                "client_version": -268364752
            },
            "recordNumber": 1
        }
    ]
}
```

VERROUILLE PAR

VERROUILLE PAR ({laTable ;} process ; utilisateur4D ; utilisateurSession ; nomProcess)

Paramètre	Type		Description
laTable	Table	⇒	Table de l'enregistrement verrouillé ou Table par défaut si ce paramètre est omis
process	Entier long	⇐	Numéro du process
utilisateur4D	Chaîne	⇐	Nom de l'utilisateur 4D
utilisateurSession	Chaîne	⇐	Nom de l'utilisateur ayant ouvert la session de travail
nomProcess	Chaîne	⇐	Nom du process

Description

VERROUILLE PAR retourne des informations sur l'utilisateur et le process qui ont verrouillé l'enregistrement. Le numéro du process(*), le nom de l'utilisateur dans l'application 4D et dans le système ainsi que le nom du process sont retournés dans les variables *process*, *utilisateur4D*, *utilisateurSession* et *nomProcess*. Vous pouvez utiliser ces informations dans une boîte de dialogue pour avertir l'utilisateur lorsqu'un enregistrement est verrouillé.

(*) Il s'agit du numéro du process sur la machine où est exécuté le code à l'origine du verrouillage. Dans le cas d'un trigger ou d'une méthode exécutée sur serveur, c'est le numéro du process "jumeau" sur le serveur qui est retourné. Dans le cas d'un process exécuté sur une machine distante, c'est le numéro du process sur la machine distante qui est retourné.

Si l'enregistrement n'est pas verrouillé, *process* prend la valeur 0 et *utilisateur4D*, *utilisateurSession* et *nomProcess* retournent des chaînes vides. Si vous essayez de charger en lecture/écriture un enregistrement qui a été supprimé, *process* retourne -1 et *utilisateur4D*, *utilisateurSession* et *nomProcess* retournent des chaînes vides.

Le paramètre *utilisateur4D* est le nom de l'utilisateur défini dans l'éditeur de mots de passe de 4D. Si aucun mot de passe n'a été défini, "Super_Utilisateur" est retourné.

Le paramètre *utilisateurSession* retourné correspond au nom de l'utilisateur ayant ouvert la session sur le poste client (ce nom est notamment affiché dans la fenêtre d'administration de 4D Server pour chaque process ouvert).

Ensembles

Présentation des ensembles

Les ensembles sont un moyen puissant et rapide de manipuler des sélections d'enregistrements. En plus de la possibilité de créer des ensembles, de les relier à la sélection courante, de les stocker, de les charger et de les effacer, 4D permet d'effectuer trois opérations standard sur les ensembles :

- Intersection,
- Union,
- Différence.

Ensembles et sélection courante

Un ensemble est une représentation d'une sélection d'enregistrements. L'idée d'ensemble est intimement liée à celle de sélection courante. Les ensembles sont généralement utilisés pour les raisons suivantes :

- Sauvegarder et ensuite restaurer une sélection lorsque l'ordre n'a pas d'importance,
- Accéder à la sélection que l'utilisateur a faite à l'écran (l'ensemble UserSet),
- Réaliser une opération logique entre des sélections.

La sélection courante est une liste de références qui pointent vers chaque enregistrement actuellement sélectionné. La liste existe en mémoire. Seuls les enregistrements sélectionnés sont dans la liste. Une sélection ne contient pas en réalité les enregistrements, mais seulement une liste de références à ces enregistrements. Chaque référence à un enregistrement utilise 4 octets en mémoire. Lorsque vous travaillez sur une table, vous travaillez toujours avec les enregistrements de la sélection courante. Lorsqu'une sélection est triée, seule la liste des références est réorganisée. Il n'y a qu'une sélection courante pour chaque table dans un process.

Comme une sélection courante, un ensemble représente une sélection d'enregistrements. Un ensemble le fait en utilisant une représentation très compacte pour chaque enregistrement. En effet, chacun est représenté par un bit (un huitième d'octet). Les opérations utilisant les ensembles sont très rapides parce les ordinateurs accomplissent très rapidement les opérations sur les bits. Un ensemble contient un bit pour chaque enregistrement de la table, que l'enregistrement soit inclus dans l'ensemble ou non. En fait, chaque bit est égal à 1 ou 0 — tout dépend si l'enregistrement est dans l'ensemble ou non.

Les ensembles sont très économiques en termes de mémoire RAM. La taille d'un ensemble, en octets, est toujours égale au nombre total des enregistrements dans la table divisé par huit. Ainsi, pour une table contenant 10 000 enregistrements, l'ensemble prend 1250 octets, ce qui fait environ 1,2 Ko de RAM.

Il peut exister plusieurs ensembles pour chaque table. En fait, les ensembles peuvent être sauvegardés sur disque indépendamment de la base. Pour modifier un enregistrement appartenant à un ensemble, vous devez d'abord utiliser l'ensemble comme sélection courante, puis modifier l'enregistrement.

Un ensemble n'est jamais trié — les enregistrements sont simplement marqués comme appartenant ou non à l'ensemble. En revanche, une sélection temporaire est triée, mais requiert davantage de mémoire dans la plupart des cas. Pour plus d'information sur les sélections temporaires, reportez-vous à la section **Présentation des Sélections Temporaires**.

Au moment de sa création, un ensemble “mémorise” l'enregistrement courant de la sélection.

Comparaison	Sélection courante	Ensembles
Nombre par table	1	illimité
Triable	Oui	Non
Sauvegardable sur disque	Non	Oui
RAM par enreg. (en octets)	Nb enreg. sélec*4	Nb total enreg./8
Combinable	Non	Oui
Contient enreg. courant	Oui	Oui, celui du moment de création

L'ensemble que vous créez appartient à la table dans laquelle il a été créé. Les opérations sur les ensembles ne peuvent être effectuées qu'entre ensembles appartenant à la même table.

Les ensembles sont indépendants des données, ce qui signifie qu'après des modifications dans une table, un ensemble peut n'être plus exact. Bien des opérations peuvent rendre un ensemble inexact. Si vous créez un ensemble de tous les habitants de New York et changez ensuite les données de l'un des enregistrements par “New Jersey,” l'ensemble ne change pas puisqu'il est simplement la représentation d'une sélection d'enregistrements. L'ajout ou la suppression d'enregistrements peut également rendre un ensemble obsolète, de même que le compactage des données. Les ensembles ne sont exacts que tant que la sélection d'origine n'est pas modifiée.

Ensembles process et interprocess

Vous pouvez utiliser trois types d'ensembles :

- **Ensembles process** : Un ensemble process n'est accessible que par le process dans lequel il a été créé. L'ensemble système **LockedSet** est un ensemble process. Les ensembles process sont effacés dès que la méthode du process est achevée. Les ensembles process ne requièrent pas de préfixe pour leur nom.

- **Ensembles interprocess** : Un ensemble est interprocess lorsque son nom est précédé des symboles (<>) — le signe “inférieur à” suivi du signe “supérieur à”. Cette syntaxe est utilisable à la fois sous Windows et Mac OS. Sous Mac OS, vous pouvez aussi utiliser le symbole “diamant” (Option+v sur un clavier français).
  Un ensemble interprocess est “visible” par tous les process de la base.
  En client/serveur, un ensemble interprocess est “visible” par tous les process du poste sur lequel il a été créé (client ou serveur).
  Le nom d'un ensemble interprocess doit être unique dans la base.

- **Ensembles locaux/Ensembles client** : Les ensembles locaux/client sont principalement destinés au mode client/serveur. Leur nom est toujours précédé du symbole dollar ($) — à l'exception de l'ensemble système **UserSet**. A la différence des autres types d'ensembles, un ensemble local/client est stocké sur le poste client.

Notes :

- La taille maximale d'un nom d'ensemble est de 255 caractères (hors symboles <> et $).
- Pour plus d'informations sur l'utilisation des ensembles en client/serveur, reportez-vous à la section **4D Server, ensembles et sélections** dans le

Visibilité des ensembles

Le tableau suivant indique les principes de visibilité des ensembles en fonction de leur portée et de leur lieu de création :

	Process client	Autres process du client	Autres clients	Process serveur	Autres process du serveur
Création dans un process client					
$test	x				
test	x			x (Trigger)	
<>test	x	x			
Création dans un process serveur					
$test				x	
test				x	
<>test				x	x

Ensembles et transactions

Un ensemble peut être créé à l'intérieur d'une transaction. Il est donc possible de définir un "ensemble des enregistrements créés pendant la transaction" et un "ensemble des enregistrements créés ou modifiés en-dehors de la transaction". Une fois la transaction terminée, l'ensemble créé pendant la transaction doit être effacé car il pourrait ne plus être une représentation exacte des enregistrements, surtout si la transaction a été annulée.

Exemple d'ensemble

Cet exemple détruit des enregistrements doublons dans une table. La table contient des informations sur des personnes. Une structure répétitive **Boucle...Fin de boucle** parcourt tous les enregistrements, comparant l'enregistrement courant à l'enregistrement précédent. Si le nom, l'adresse et le code postal sont identiques, l'enregistrement est ajouté à un ensemble. A la fin de la boucle, l'ensemble devient la sélection courante et l'ancienne sélection courante est détruite :

```
ENSEMBLE VIDE([Personnes];"Doublons")
 // Créer un ensemble vide pour les doublons
TOUT SELECTIONNER([Personnes])
 // Tous les enregistrements
 // Trier les enregistrements par code postal, adresse et nom pour
 // que les doublons soient les uns à côté des autres
TRIER([Personnes];[Personnes]CODEPOSTAL;>;[Personnes]Adresse;>;[Personnes]Nom;>)
 // Initialiser les variables conservant les champs des enregistrements précédents
$Nom:=[Personnes]Nom
$Adresse:=[Personnes]Adresse
$CODEPOSTAL:=[Personnes]CODEPOSTAL
 // Aller au second enregistrement pour le comparer au premier
ENREGISTREMENT SUIVANT([Personnes])
Boucle($i;2;Enregistrements dans table([Personnes]))
 // Parcourir les enregistrements en partant à 2
 // Si nom, adresse et CODEPOSTAL sont identiques au
 // précédent enregistrement, alors c'est un doublon.
   Si(([Personnes]Nom=$Nom) & ([Personnes]Adresse=$Adresse) & ([Personnes]CODEPOSTAL=$CODEPOSTAL))
 // Ajouter enregistrement (le doublon) à l'ensemble
      ADJOINDRE ELEMENT([Personnes];"Doublons")
   Sinon
 // Garder les nom, adresse et CODEPOSTAL de cet enregistrement pour comparer avec le prochain
      $Nom:=[Personnes]Nom
      $Adresse:=[Personnes]Adresse
      $CODEPOSTAL:=[Personnes]CODEPOSTAL
   Fin de si
 // Passer à l'enregistrement suivant
   ENREGISTREMENT SUIVANT([Personnes])
Fin de boucle
 // Utiliser doublons trouvés (en tant que sélection courante)
UTILISER ENSEMBLE("Doublons")
 // Détruire doublons
SUPPRIMER SELECTION([Personnes])
 // Supprimer l'ensemble de la mémoire
EFFACER ENSEMBLE("Doublons")
```

Au lieu de supprimer immédiatement les enregistrements à la fin de la méthode, vous pouvez les afficher à l'écran ou les imprimer si des comparaisons plus fines doivent être menées.

Ensemble système UserSet

4D gère un ensemble système local/client nommé *UserSet*. *UserSet* contient automatiquement l'ensemble des derniers enregistrements marqués (“surlignés”) à l'écran par l'utilisateur dans un formulaire en liste. Ainsi, vous pouvez afficher un groupe d'enregistrements avec **MODIFIER SELECTION** ou **VISUALISER SELECTION** , demander à l'utilisateur d'en sélectionner certains et retourner le résultat de cette sélection manuelle dans un ensemble que vous nommez ou dans une sélection.

Il existe un seul *UserSet* par process. Les tables ne disposent pas de leur propre *UserSet*. Le *UserSet* n'est associé à une table que lorsqu'une sélection d'enregistrements est affichée pour cette table.

4D gère l'ensemble *UserSet* pour les formulaires liste affichés en mode Développement et via les commandes **MODIFIER SELECTION** ou **VISUALISER SELECTION**. En revanche, ce mécanisme n'est pas actif pour les sous-formulaires.

La méthode ci-dessous illustre comment afficher des enregistrements pour permettre à l'utilisateur d'en sélectionner quelques-uns, et ensuite utiliser le *UserSet* pour afficher les enregistrements sélectionnés :

```
   ` Afficher tous les enregistrements et permettre à l'utilisateur d'en sélectionner un certain nombre.
   ` Puis afficher cette sélection en utilisant UserSet pour modifier la sélection courante.
FORM FIXER SORTIE([Personnes];"Display") ` Choisir le formulaire sortie
TOUT SELECTIONNER([Personnes]) ` Sélection de toutes les personnes
ALERTE("Appuyer Ctrl ou Commande + Clic pour sélectionner des enregistrements.")
VISUALISER SELECTION([Personnes]) ` Afficher les personnes
UTILISER ENSEMBLE("UserSet") ` Utiliser les personnes sélectionnées
ALERTE("Vous avez choisi les personnes suivantes.")
VISUALISER SELECTION([Personnes]) ` Afficher les personnes sélectionnées
```

4D Server : Bien que son nom ne débute pas par le caractère "$", l'ensemble système *UserSet* est un ensemble client. Par conséquent, lors de l'utilisation des commandes **INTERSECTION**, **REUNION** et **DIFFERENCE**, veillez à ne comparer *UserSet* qu'à d'autres ensembles clients. Pour plus d'informations, reportez-vous aux descriptions de ces commandes ainsi qu'à la section **4D Server, ensembles et sélections** dans le Guide de référence de 4D Server.

Ensemble système LockedSet

Les commandes **APPLIQUER A SELECTION**, **SUPPRIMER SELECTION**, **TABLEAU VERS SELECTION** et **JSON VERS SELECTION** créent un ensemble système nommé **LockedSet** lorsqu'elles sont utilisées en environnement multiprocess.
Les commandes de recherche créent également un ensemble système *LockedSet* lorsqu'elles trouvent des enregistrements verrouillés dans le contexte de 'recherche et verrouillage' (cf. commande **FIXER RECHERCHE ET VERROUILLAGE**).
LockedSet indique quels enregistrements étaient verrouillés lors de l'exécution d'une commande.

ADJOINDRE ELEMENT

ADJOINDRE ELEMENT ({laTable ;} ensemble)

Paramètre	Type		Description
laTable	Table	⇨	Table de l'enregistrement courant ou Table par défaut si ce paramètre est omis
ensemble	Chaîne	⇨	Nom de l'ensemble auquel ajouter l'enregistrement courant

Description

ADJOINDRE ELEMENT ajoute l'enregistrement courant de *laTable* à *ensemble*. L'ensemble doit avoir déjà été créé ; si *ensemble* n'existe pas, une erreur est retournée. S'il n'y a pas d'enregistrement courant pour *laTable* , **ADJOINDRE ELEMENT** ne fait rien.

Appartient a ensemble

Appartient a ensemble (ensemble) -> Résultat

Paramètre	Type		Description
ensemble	Chaîne	⇨	Nom de l'ensemble à tester
Résultat	Booléen	⮌	L'enregistrement courant est dans l'ensemble (Vrai) ou l'enregistrement courant n'est pas dans l'ensemble (Faux)

Description

Appartient a ensemble teste si l'enregistrement courant de la table est inclus dans *ensemble*. La fonction **Appartient a ensemble** retourne Vrai si l'enregistrement courant de la table est dans *ensemble*, et retourne Faux si l'enregistrement courant de la table n'est pas dans *ensemble*.

Exemple

L'exemple suivant est la méthode objet d'un bouton testant si l'enregistrement courant est inclus dans l'ensemble des meilleurs clients :

```
Si(Appartient a ensemble("Meilleurs"))
   ALERTE("C'est un de nos meilleurs clients.")
Sinon
   ALERTE("Ce n'est pas un de nos meilleurs clients.")
Fin de si
```

CHARGER ENSEMBLE

CHARGER ENSEMBLE ({laTable ;} ensemble ; nomFichier)

Paramètre	Type		Description
laTable	Table	⇒	Table à laquelle appartient l'ensemble ou Table par défaut si ce paramètre est omis
ensemble	Chaîne	⇒	Nom de l'ensemble à créer en mémoire
nomFichier	Chaîne	⇒	Document disque contenant l'ensemble

Description

CHARGER ENSEMBLE charge un ensemble depuis le fichier *nomFichier*, créé à l'aide de la commande **STOCKER ENSEMBLE**.

L'ensemble stocké dans *nomFichier* doit s'appliquer à *laTable*. Si *ensemble* existait déjà en mémoire, il est réécrit.

Le paramètre *nomFichier* est le nom du fichier disque contenant l'ensemble. Il n'est pas nécessaire que ce fichier ait le même nom que l'ensemble. Si vous passez une chaîne vide dans *nomFichier*, une boîte de dialogue standard d'ouverture de fichiers s'affiche, permettant à l'utilisateur de choisir l'ensemble à charger.

ATTENTION : Rappelez-vous qu'un ensemble est l'image d'une sélection d'enregistrements au moment précis où l'ensemble est créé. Si les enregistrements représentés par l'ensemble sont modifiés, celui-ci devient obsolète. En conséquence, vous devez stocker et charger des ensembles avec des enregistrements dont le contenu varie peu. De multiples événements peuvent rendre un ensemble obsolète : modification ou suppression d'un enregistrement de l'ensemble, ou encore modification des critères ayant déterminé la création de l'ensemble.

Exemple

L'exemple suivant utilise **CHARGER ENSEMBLE** pour charger l'ensemble des locaux de l'entreprise Dupont SARL à Paris :

```
  ` Charger l'ensemble en mémoire
CHARGER ENSEMBLE([Entreprises];"Paris Dupont SARL";"PaDupontEns")
UTILISER ENSEMBLE("Paris Dupont SARL") ` Modifier la sélection courante avec l'ensemble
EFFACER ENSEMBLE("Paris Dupont SARL") ` Effacer l'ensemble de la mémoire
```

Variables et ensembles système

Si l'utilisateur clique sur Annuler dans la boîte de dialogue d'ouverture de fichiers, ou si une erreur se produit pendant le chargement, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.

COPIER ENSEMBLE

COPIER ENSEMBLE (srcEns ; dstEns)

Paramètre	Type		Description
srcEns	Chaîne	➡	Nom de l'ensemble source
dstEns	Chaîne	➡	Nom de l'ensemble de destination

Description

La commande **COPIER ENSEMBLE** copie le contenu de l'ensemble *srcEns* dans l'ensemble *dstEns*.

Chaque ensemble peut être de type process, interprocess ou local/client. Les deux ensembles peuvent être de type différent (comme dans les exemples ci-dessous), pourvu qu'ils soient visibles sur le poste. Pour plus d'informations sur ce point, reportez-vous au paragraphe "**Visibilité des ensembles**".

Exemple 1

L'exemple suivant, en client/serveur, copie l'ensemble local "*$SetA"*, conservé sur le poste client, vers l'ensemble process *"SetB"*, conservé sur le poste serveur :

```
COPIER ENSEMBLE("$SetA";"SetB")
```

Exemple 2

L'exemple suivant, en client/serveur, copie l'ensemble process "*SetA*", conservé sur le poste serveur, vers l'ensemble local *"$SetB"*, conservé sur le poste client :

```
COPIER ENSEMBLE("SetA";"$SetB")
```

CREER ENSEMBLE SUR TABLEAU

CREER ENSEMBLE SUR TABLEAU (laTable ; tabEnrg {; nomEnsemble})

Paramètre	Type		Description
laTable	Table	⇒	Table de l'ensemble
tabEnrg	Entier long, Tableau booléen	⇒	Tableau de n° d'enregistrements, ou Tableau de booléens (Vrai = l'enregistrement est dans l'ensemble, Faux = il n'est pas dans l'ensemble)
nomEnsemble	Chaîne	⇒	Nom de l'ensemble à créer, ou Appliquer la commande à l'ensemble Userset si ce paramètre est omis ou vide

Description

La commande **CREER ENSEMBLE SUR TABLEAU** crée l'ensemble *nomEnsemble* à partir :

- soit du tableau de numéros d'enregistrements absolus *tabEnrg* de *laTable*,
- soit du tableau de booléens *tabEnrg* ; dans ce cas, les valeurs du tableau indiquent l'appartenance (VRAI) ou non (FAUX) de chaque enregistrement de table à l'ensemble *nomEnsemble*.

Lorsque vous utilisez la commande avec un tableau d'entiers longs, tous les numéros du tableau représentent la liste des numéros d'enregistrements qui feront partie de l'ensemble *nomEnsemble*. Si un numéro est invalide (enregistrement non créé), l'erreur -10503 est générée.

Lorsque vous utilisez la commande avec un tableau de booléens, le Nième élément du tableau représente l'intégration (VRAI) ou non (FAUX) de l'enregistrement numéro N à l'ensemble *nomEnsemble*. En principe, le nombre d'éléments du tableau doit être égal au nombre d'enregistrements de *table*. Si le tableau est plus petit que le nombre d'enregistrements, seuls les enregistrements définis par le tableau pourront faire partie de l'ensemble.

Note : Avec un tableau de booléens, la commande utilise les éléments à partir du numéro 0 jusqu'au numéro N-1.

Si vous ne passez pas le paramètre *nomEnsemble* ou si vous passez une chaîne vide, la commande s'applique à l'ensemble système *Userset*.

Gestion des erreurs

Dans un tableau d'entier longs, si un numéro d'enregistrement est invalide (enregistrement non créé), l'erreur -10503 est générée.

DIFFERENCE

DIFFERENCE (ensemble1 ; ensemble2 ; résultat)

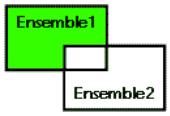
Paramètre	Type		Description
ensemble1	Chaîne	➡	Ensemble initial
ensemble2	Chaîne	➡	Ensemble à exclure
résultat	Chaîne	➡	Ensemble résultant

Description

DIFFERENCE fusionne *ensemble1* et *ensemble2* et exclut de l'ensemble *résultat* tous les enregistrements se trouvant dans *ensemble2*. Autrement dit, un enregistrement est inclus dans l'ensemble *résultat* s'il appartient à *ensemble1* mais n'appartient pas à *ensemble2*. Le tableau suivant liste les résultats possibles d'une opération de différence d'ensembles.

Ensemble1	Ensemble2	Ensemble résultant
Oui	Non	Oui
Oui	Oui	Non
Non	Oui	Non
Non	Non	Non

Le schéma ci-dessous est la représentation graphique d'une opération de différence entre deux ensembles. La zone colorée est l'ensemble résultant.



L'ensemble *résultat* est créé par **DIFFERENCE**. Il remplace tout ensemble du même nom existant déjà, y compris *ensemble1* et *ensemble2*. Les ensembles *ensemble1* et *ensemble2* doivent appartenir à la même table. L'ensemble *résultat* appartient à la même table que *ensemble1* et *ensemble2*.

4D Server : En mode client/serveur, les ensembles sont "visibles" en fonction de leur type (interprocess, process et local) et de leur lieu de création (serveur ou client). **DIFFERENCE** requiert que les trois ensembles soient visibles sur la même machine. Pour plus d'informations sur ce point, reportez-vous au paragraphe **4D Server, ensembles et sélections** dans le manuel de référence de 4D Server.

Exemple

L'exemple suivant exclut les enregistrements sélectionnés par l'utilisateur. Les enregistrements sont affichés à l'écran par l'instruction suivante :

```
VISUALISER SELECTION([Clients]) ` Affichage des clients sous forme de liste
```

Un bouton associé à une méthode objet est placé en bas de la liste. La méthode objet exclut les enregistrements sélectionnés par l'utilisateur (l'ensemble système nommé **UserSet**) et affiche une sélection réduite :

```
NOMMER ENSEMBLE([Clients];"$Courant") ` Création d'un ensemble à partir de la sélection courante
DIFFERENCE("$Courant";"UserSet";"$Courant") ` Exclusion des enregistrements sélectionnés
UTILISER ENSEMBLE("$Courant") ` Utilisation du nouvel ensemble
EFFACER ENSEMBLE("$Courant") ` Effacement de l'ensemble
```

EFFACER ENSEMBLE

EFFACER ENSEMBLE (ensemble)

Paramètre	Type		Description
ensemble	Chaîne	⇨	Nom de l'ensemble à effacer de la mémoire

Description

EFFACER ENSEMBLE efface *ensemble* de la mémoire et la libère ainsi pour d'autres utilisations. **EFFACER ENSEMBLE** n'a aucune conséquence sur les tables, sélections ou enregistrements. Pour sauvegarder un ensemble avant de l'effacer, utiliser la commande **STOCKER ENSEMBLE**. Comme les ensembles consomment de la mémoire, pensez à les effacer dès qu'ils ne sont plus nécessaires.

Exemple

Reportez-vous à l'exemple de la commande **UTILISER ENSEMBLE**.

ENLEVER ELEMENT

ENLEVER ELEMENT ({laTable ;} ensemble)

Paramètre	Type		Description
laTable	Table	⇨	Table de l'enregistrement courant ou Table par défaut si ce paramètre est omis
ensemble	Chaîne	⇨	Nom de l'ensemble duquel supprimer l'enregistrement courant

Description

ENLEVER ELEMENT supprime l'enregistrement courant de *laTable* de l'ensemble *ensemble*. L'ensemble doit déjà exister ; s'il n'existe pas, une erreur est générée. S'il n'y a pas d'enregistrement courant dans *laTable*, **ENLEVER ELEMENT** ne fait rien.

Enregistrements dans ensemble

Enregistrements dans ensemble (ensemble) -> Résultat

Paramètre	Type		Description
ensemble	Chaîne	⇒	Nom de l'ensemble à tester
Résultat	Entier long	⇐	Nombre d'enregistrements dans l'ensemble

Description

Enregistrements dans ensemble retourne le nombre d'enregistrements présents dans *ensemble*. Si *ensemble* n'existe pas ou s'il n'y a pas d'enregistrements dans *ensemble*, **Enregistrements dans ensemble** retourne 0.

Exemple

L'exemple suivant affiche une boîte de dialogue d'alerte qui indique le pourcentage des clients qui sont considérés comme les meillleurs :

```
  ` Calculer d'abord le pourcentage
$Pourcent :=(Enregistrements dans ensemble("Meilleurs")/Enregistrements dans table([Clients]))*100
  ` Afficher une alerte avec le pourcentage
ALERTE(Chaine($Pourcent;"##0%")+" de nos clients sont nos meilleurs clients.")
```

ENSEMBLE VIDE

ENSEMBLE VIDE ({laTable ;} ensemble)

Paramètre	Type		Description
laTable	Table	⇒	Table pour laquelle créer un ensemble vide ou Table par défaut si ce paramètre est omis
ensemble	Chaîne	⇒	Nom du nouvel ensemble vide

Description

ENSEMBLE VIDE crée un ensemble vide, *ensemble*, pour *laTable*. Vous pouvez ajouter des enregistrements dans cet ensemble à l'aide de la commande **ADJOINDRE ELEMENT**. Si un ensemble du même nom existe déjà, il est effacé et remplacé par le nouvel ensemble.

Note : Il n'est pas indispensable d'appeler la commande **ENSEMBLE VIDE** avant d'utiliser la commande **NOMMER ENSEMBLE**.

Exemple

Reportez-vous à l'exemple proposé dans la section **Présentation des ensembles**.

INTERSECTION

INTERSECTION (ensemble1 ; ensemble2 ; résultat)

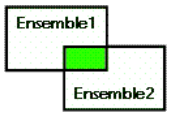
Paramètre	Type		Description
ensemble1	Chaîne	➡	Premier ensemble
ensemble2	Chaîne	➡	Second ensemble
résultat	Chaîne	➡	Ensemble résultant

Description

INTERSECTION compare *ensemble1* et *ensemble2* et sélectionne uniquement les enregistrements se trouvant à la fois dans *ensemble1* et dans *ensemble2*. Le tableau suivant liste les résultats possibles d'une opération d'intersection d'ensembles.

Ensemble1	Ensemble2	Ensemble résultant
Oui	Non	Non
Oui	Oui	Oui
Non	Oui	Non
Non	Non	Non

Le schéma ci-dessous est la représentation graphique de l'intersection de deux ensembles. La zone colorée est l'ensemble résultant.



L'ensemble *résultat* est créé par **INTERSECTION**. Il remplace tout ensemble du même nom existant déjà, y compris *ensemble1* et *ensemble2*. Les ensembles de départ *ensemble1* et *ensemble2* doivent appartenir à la même table. L'ensemble *résultat* appartient à la même table que *ensemble1* et *ensemble2*. Si le même enregistrement courant était défini dans *ensemble1* et *ensemble2*, il reste mémorisé dans l'ensemble *résultat*. Sinon, l'ensemble *résultat* ne comporte pas d'enregistrement courant.

4D Server : En mode client/serveur, les ensembles sont "visibles" en fonction de leur type (interprocess, process et local) et de leur lieu de création (serveur ou client). **INTERSECTION** requiert que les trois ensembles soient visibles sur la même machine. Pour plus d'informations sur ce point, reportez-vous à la section **4D Server, ensembles et sélections** dans le manuel de référence de 4D Server.

Exemple

L'exemple suivant recherche les clients en contact avec deux représentants, Jean et Grégoire. Chaque représentant dispose d'un ensemble regroupant ses clients. Les clients se trouvant dans les deux ensembles sont en contact avec Jean et Grégoire :

```
INTERSECTION("Jean";"Grégoire";"Doublon") ` Doublon reçoit les clients appartenant aux 2 ensembles
UTILISER ENSEMBLE("Doublon") ` Modification de la sélection courante
EFFACER ENSEMBLE("Doublon") ` Effacement de cet ensemble mais sauvegarde des autres
VISUALISER SELECTION([Clients]) ` Affichage des clients en contact avec les deux commerciaux
```

NOMMER ENSEMBLE

NOMMER ENSEMBLE ({laTable ;} ensemble)

Paramètre	Type		Description
laTable	Table	⇨	Table pour laquelle vous voulez créer un ensemble à partir de la sélection courante ou Table par défaut si ce paramètre est omis
ensemble	Chaîne	⇨	Nom du nouvel ensemble

Description

NOMMER ENSEMBLE crée un nouvel ensemble, *ensemble*, pour *laTable*, et y place la sélection courante. Le pointeur d'enregistrement courant de la table est sauvegardé avec *ensemble*. Si *ensemble* est passé à la commande **UTILISER ENSEMBLE**, la sélection courante et l'enregistrement courant sont restitués. Comme pour tout ensemble, il ne peut y avoir de tri, et lorsque *ensemble* est appelé, l'ordre par défaut est utilisé. Si un ensemble du même nom existe déjà, il est effacé et remplacé par le nouvel ensemble.

Exemple

L'exemple suivant crée un ensemble après qu'une recherche ait été effectuée, de manière à ce que l'ensemble puisse être stocké sur disque :

```
CHERCHER([Personnes]) ` L'utilisateur effectue une recherche
NOMMER ENSEMBLE([Personnes];"EnsembleRecherche") ` Création d'un nouvel ensemble
STOCKER ENSEMBLE("EnsembleRecherche";"MaRecherche") ` L'ensemble est stocké sur disque
```

REUNION

REUNION (ensemble1 ; ensemble2 ; résultat)

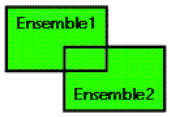
Paramètre	Type		Description
ensemble1	Chaîne	➡	Premier ensemble
ensemble2	Chaîne	➡	Second ensemble
résultat	Chaîne	➡	Ensemble résultant

Description

REUNION crée un nouvel ensemble contenant tous les enregistrements de *ensemble1* et *ensemble2*. Le tableau suivant liste les résultats possibles d'une opération de réunion d'ensembles.

Ensemble1	Ensemble2	Ensemble résultant
Oui	Non	Oui
Oui	Oui	Oui
Non	Oui	Oui
Non	Non	Non

Le schéma ci-dessous est la représentation graphique de la réunion de deux ensembles. La zone colorée est l'ensemble résultant.



L'ensemble *résultat* est créé par **REUNION**. Il remplace tout ensemble du même nom existant déjà, y compris *ensemble1* et *ensemble2*. Les ensembles de départ *ensemble1* et *ensemble2* doivent appartenir à la même table. L'ensemble *résultat* appartient à la même table que *ensemble1* et *ensemble2*. L'enregistrement courant de *résultat* est celui de *ensemble1*.

4D Server : En mode client/serveur, les ensembles sont "visibles" en fonction de leur type (interprocess, process et local) et de leur lieu de création (serveur ou client). **REUNION** requiert que les trois ensembles soient visibles sur la même machine. Pour plus d'informations sur ce point, reportez-vous au paragraphe **4D Server, ensembles et sélections** dans le manuel de référence de 4D Server.

Exemple

L'exemple suivant ajoute des enregistrements à l'ensemble des meilleurs clients. Les enregistrements sont affichés à l'écran. Ensuite, l'ensemble des meilleurs clients est chargé du disque, et tous les enregistrements sélectionnés par l'utilisateur (l'ensemble système **UserSet**) sont ajoutés. Enfin, le nouvel ensemble est sauvegardé sur le disque :

```
TOUT SELECTIONNER([Clients]) ` Sélection de tous les enregistrements
VISUALISER SELECTION([Clients]) ` Afficher tous les clients en mode liste
CHARGER ENSEMBLE("$Meilleurs";"$Meilleurs.sav") ` Chargement de l'ensemble des meilleurs clients
REUNION("$Meilleurs";"UserSet";"$Meilleurs") ` Ajout de toute sélection à l'ensemble
STOCKER ENSEMBLE("$Meilleurs";"$Meilleurs.sav") ` Sauvegarde de l'ensemble des meilleurs clients
```

STOCKER ENSEMBLE

STOCKER ENSEMBLE (ensemble ; nomFichier)

Paramètre	Type		Description
ensemble	Chaîne	⇒	Nom de l'ensemble à stocker
nomFichier	Chaîne	⇒	Nom du fichier dans lequel stocker l'ensemble

Description

STOCKER ENSEMBLE sauvegarde *ensemble* dans le fichier disque *document*.

Il n'est pas nécessaire que *document* ait le même nom que l'ensemble. Si vous passez une chaîne vide dans *document*, une boîte de dialogue standard de sauvegarde de fichiers apparaît, permettant à l'utilisateur de saisir un nom de fichier. Vous pourrez utiliser la commande **CHARGER ENSEMBLE** pour charger un ensemble stocké sur disque.

Si l'utilisateur clique sur le bouton Annuler dans la boîte de dialogue de sauvegarde de fichiers, ou si une erreur se produit lors de la sauvegarde, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.

La commande **STOCKER ENSEMBLE** est souvent utilisée pour stocker sur disque les résultats d'une recherche particulièrement longue.

ATTENTION : Rappelez-vous qu'un ensemble est l'image d'une sélection d'enregistrements au moment précis où l'ensemble est créé. Si les enregistrements représentés par l'ensemble sont modifiés, celui-ci devient obsolète. En conséquence, vous devez créer et sauvegarder des ensembles représentant des enregistrements dont le contenu varie peu. De multiples événements peuvent rendre un ensemble obsolète : modification ou suppression d'un enregistrement de l'ensemble, ou encore modification des critères ayant déterminé la création de l'ensemble. Rappelez-vous également que les ensembles ne stockent pas les valeurs des champs.

Exemple

L'exemple suivant affiche la boîte de dialogue standard d'enregistrement de fichiers afin de permettre à l'utilisateur de saisir le nom du fichier contenant l'ensemble :

```
STOCKER ENSEMBLE("UnEnsemble";"")
```

Variables et ensembles système

Si l'utilisateur clique sur le bouton Annuler dans la boîte de dialogue standard de sauvegarde de documents, ou si une erreur se produit pendant la sauvegarde, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.

UTILISER ENSEMBLE

UTILISER ENSEMBLE (ensemble)

Paramètre	Type		Description
ensemble	Chaîne	⇨	Nom de l'ensemble à utiliser

Description

UTILISER ENSEMBLE crée, avec les enregistrements de *ensemble*, une nouvelle sélection courante pour la table à laquelle *ensemble* appartient.

Au moment où vous créez un ensemble, la position de l'enregistrement courant est sauvegardée. **UTILISER ENSEMBLE** récupère cette information et fait de l'enregistrement le nouvel enregistrement courant. Si vous supprimez cet enregistrement avant d'exécuter **UTILISER ENSEMBLE**, 4D sélectionne comme enregistrement courant le premier enregistrement de l'ensemble. Les commandes du thème "Ensembles", **REUNION**, **INTERSECTION**, **DIFFERENCE** et **ADJOINDRE ELEMENT** réinitialisent l'enregistrement courant.

Si vous avez créé un ensemble ne contenant pas de position d'enregistrement courant, **UTILISER ENSEMBLE** désigne le premier enregistrement de l'ensemble comme enregistrement courant.

ATTENTION : Rappelez-vous qu'un ensemble est la représentation d'une sélection d'enregistrements à un instant donné (au moment de la création de l'ensemble). Si les enregistrements que l'ensemble représente sont modifiés, il se peut que celui-ci ne soit plus valide. En conséquence, un ensemble sauvegardé sur disque doit généralement représenter un groupe d'enregistrements qui ne change pas souvent. De multiples événements peuvent rendre un ensemble invalide, comme par exemple la suppression ou la modification d'un enregistrement de l'ensemble, ou encore la modification des critères de création de l'ensemble.

Exemple

L'exemple suivant utilise **CHARGER ENSEMBLE** pour charger un ensemble des sites de la société Dubois à Paris. **UTILISER ENSEMBLE** est ensuite appelée pour faire de l'ensemble la sélection courante :

```
` Charger l'ensemble en mémoire
CHARGER ENSEMBLE([Entreprises];"DuboisParis";"ENSDuboisParis")
UTILISER ENSEMBLE("DuboisParis") ` Modification de la sélection courante
EFFACER ENSEMBLE("DuboisParis") ` Effacement de l'ensemble de la mémoire
```

Environnement 4D

- Compacter fichier donnees
- CREER FICHIER DONNEES
- Dossier 4D Modifié 16.0
- Fichier 4D Nouveauté 16.0
- Fichier application
- Fichier donnees
- Fichier donnees verrouille
- Fichier structure
- FIXER DOSSIER MISE A JOUR
- FIXER LANGUE BASE
- FIXER PARAMETRE BASE
- GENERER APPLICATION
- Lire chemin journal derniere mise a jour
- Lire fragmentation table
- LIRE INFORMATIONS SERIALISATION
- Lire langue base
- Lire mesures base
- Lire parametre base
- LISTE COMPOSANTS
- LISTE PLUGINS
- Mode compile
- NOTIFIER MODIFICATION DOSSIER RESOURCES
- OUVRIR BASE
- OUVRIR CENTRE DE SECURITE
- OUVRIR FENETRE ADMINISTRATION
- OUVRIR FENETRE PROPRIETES Modifié 16.0
- OUVRIR FICHIER DONNEES
- QUITTER 4D
- REDEMARRER 4D
- Type application
- Type version
- VERIFIER FICHIER DONNEES
- VERIFIER FICHIER DONNEES OUVERT
- Version application
- *_o_AJOUTER SEGMENT DE DONNÉES*
- *_o_LISTE SEGMENTS DE DONNÉES*

Compacter fichier donnees

Compacter fichier donnees (cheminStructure ; cheminDonnées {; dossierArchive {; options {; méthode}}}) -> Résultat

Paramètre	Type		Description
cheminStructure	Texte	⇒	Chemin d'accès du fichier de structure
cheminDonnées	Texte	⇒	Chemin d'accès du fichier de données
dossierArchive	Texte	⇒	Chemin d'accès du dossier dans lequel placer le fichier de données original
options	Entier long	⇒	Options de compactage
méthode	Texte	⇒	Nom de la méthode 4D de rétro-appel
Résultat	Texte	⇐	Chemin d'accès complet du dossier contenant le fichier de données original

Description

La commande **Compacter fichier donnees** effectue un compactage du fichier de données désigné par le paramètre *cheminDonnées* associé au fichier de structure *cheminStructure*. Pour plus d'informations sur le compactage, reportez-vous au manuel Mode Développement.

Pour assurer la continuité du fonctionnement de la base, le nouveau fichier de données compacté remplace automatiquement le fichier original. Par sécurité, le fichier original n'est pas modifié et est déplacé dans un dossier spécial nommé "Replaced files (compacting) AAAA-MM-JJ HH-MM-SS" où AAAA-MM-JJ HH-MM-SS représente la date et l'heure de la sauvegarde. Par exemple : "Replaced files (compacting) 2015-09-27 15-20-35".

La commande retourne le chemin d'accès complet du dossier effectivement créé pour stocker le fichier de données original. Cette commande peut être exécutée depuis 4D (mode local) ou 4D Server uniquement (procédure stockée). Le fichier de données à compacter doit correspondre au fichier de structure désigné par *cheminStructure*. En outre, il ne doit PAS être ouvert au moment de l'exécution de la commande, sinon une erreur est générée.
Si une erreur se produit durant le processus de compactage, les fichiers originaux sont conservés à leur emplacement initial. Si un fichier d'index (fichier .4DIndx) est associé au fichier de données, il est également compacté. Comme pour le fichier de données, le fichier original est sauvegardé et la nouvelle version compactée remplace la précédente.

- Passez dans *cheminStructure* le chemin d'accès complet du fichier de structure associé au fichier de données que vous souhaitez compacter. Cette information est nécessaire à la procédure de compactage. Le chemin d'accès doit être exprimé dans la syntaxe du système d'exploitation. Vous pouvez également passer une chaîne vide, dans ce cas une boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de désigner le fichier de structure à utiliser.
- Vous pouvez passer dans *cheminDonnées* une chaîne vide, un nom de fichier ou un chemin d'accès complet, exprimé dans la syntaxe du système d'exploitation. Si vous passez une chaîne vide, la boîte de dialogue standard d'ouverture de fichier apparaît, permettant à l'utilisateur de désigner le fichier de données à compacter. Ce fichier doit correspondre au fichier de structure défini dans le paramètre *cheminStructure*. Si vous passez uniquement un nom de fichier de données, 4D le recherchera au même niveau que le fichier de structure.
- Le paramètre facultatif *dossierArchive* permet de désigner l'emplacement du dossier "Replaced files (compacting) Dateheure" destiné à recueillir les versions originales des fichiers de données ainsi que des éventuels fichiers d'index.
  La commande retourne le chemin d'accès complet du dossier effectivement créé.
  - Si vous omettez ce paramètre, les fichiers d'origine sont automatiquement déplacés dans un dossier "Replaced files (compacting) Dateheure" créé à côté du fichier de structure.
  - Si vous passez une chaîne vide, une boîte de dialogue standard d'ouverture de dossier apparaît, permettant à l'utilisateur de désigner l'emplacement du dossier à créer.
  - Si vous passez un chemin d'accès (exprimé dans la syntaxe du système d'exploitation), la commande créera le dossier "Replaced files (compacting) Dateheure" à cet emplacement.
- Le paramètre facultatif *options* permet de définir diverses options liées au compactage. Pour cela, utilisez les constantes suivantes, placées dans le thème **Maintenance fichier de données**. Vous pouvez passer plusieurs options en les cumulant :

Constante	Type	Valeur	Comment
Compacter table adresses	Entier long	131072	Forcer la réécriture de la table d'adresses des enregistrements (ralentit le compactage). A noter que dans ce cas, les numéros des enregistrements sont réécrits. Si vous passez uniquement cette option, 4D active automatiquement l'option 'Mettre à jour enregistrements'.
Créer un process	Entier long	32768	Lorsque cette option est passée, le compactage sera asynchrone et vous devrez gérer les résultats à l'aide de la méthode de rétro-appel (voir ci-dessous). 4D n'affichera pas de barre de progression (il est possible de le faire via la méthode de rétro-appel). La variable système OK prendra la valeur 1 si le process a été correctement lancé et 0 dans les autres cas. Lorsque cette option n'est pas passée, la variable OK prendra la valeur 1 si le compactage s'est correctement déroulé et sinon 0.
Mettre à jour enregistrements	Entier long	65536	Forcer la réécriture de tous les enregistrements suivant la définition courante des champs dans la structure
Ne pas créer d'historique	Entier long	16384	En principe, la commande crée un fichier d'historique au format xml (reportez-vous à la fin de la description de la commande). Vous pouvez annuler ce fonctionnement en passant cette option.
Nom historique avec date heure	Entier long	262144	Lorsque cette option est passée, le nom du fichier d'historique généré contiendra la date et l'heure de sa création, par conséquent il ne remplacera aucun fichier d'historique éventuellement déjà généré. Par défaut, si cette option n'est pas passée, le nom du fichier n'est pas horodaté, et chaque nouveau fichier généré remplace le précédent.

- Le paramètre *méthode* permet de désigner une méthode de rétro-appel qui sera régulièrement appelée durant le compactage si l'option Créer un process a été passée. Dans le cas contraire, la méthode de rétro-appel n'est jamais appelée. Pour plus d'informations sur cette méthode, reportez-vous à la description de la commande **VERIFIER FICHIER DONNEES**.
  Si la méthode de rétro-appel n'existe pas dans la base, une erreur est générée et la variable système OK prend la valeur 0.

Par défaut, la commande **Compacter fichier donnees** crée un fichier d'historique au format xml (si vous n'avez pas passé l'option Ne pas créer d'historique, cf. paramètre *options*). Son nom est basé sur celui du fichier de structure de la base courante et il est également placé dans le dossier **Logs** de cette base. Par exemple, pour un fichier de structure nommé "myDB.4db", le fichier d'historique sera nommé "myDB_Compact_Log.xml".
Si vous avez passé l'option Nom historique avec date heure, le nom du fichier d'historique inclut la date et l'heure de sa création sous la forme "AAAA-MM-JJ HH-MM-SS", ce qui donne par exemple : "myDB_Compact_Log_2015-09-27 15-20-35.xml". Ce principe permet d'éviter que chaque nouveau fichier d'historique écrase le précédent, mais pourra nécessiter ultérieurement une action manuelle afin de supprimer les fichiers superflus.
Quelle que soit l'option sélectionnée, dès lors qu'un fichier d'historique est généré, son chemin est retourné dans la variable système *Document* à l'issue de l'exécution de la commande.

Exemple

L'exemple suivant (Windows) effectue le compactage d'un fichier de données :

```
$ficStruc:=Fichier structure
$ficDonnées:="C:\Bases\Factures\Janvier\Factures.4dd"
$ficOrig:="C:\Bases\Factures\Archives\Janvier\"
$dossierArch:=Compacter fichier donnees($ficStruc;$ficDonnées;$ficOrig)
```

Variables et ensembles système

Si l'opération de compactage s'est déroulée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0. Si un fichier d'historique a été généré, son chemin complet est retourné dans la variable système Document.

CREER FICHIER DONNEES

CREER FICHIER DONNEES (cheminAccès)

Paramètre	Type		Description
cheminAccès	Chaîne	⇨	Nom ou chemin d'accès complet du fichier de données à créer

Description

La commande **CREER FICHIER DONNEES** permet de créer un nouveau fichier de données sur disque et de remplacer à la volée le fichier de données ouvert par l'application 4D.

Le fonctionnement général de cette commande est identique à celui de la commande **OUVRIR FICHIER DONNEES**, à la différence près que le nouveau fichier de données désigné par le paramètre *cheminAccès* est créé juste après la réouverture du fichier de structure.

Avant de lancer l'opération, la commande vérifie que le chemin spécifié ne correspond pas à un fichier existant.

4D Server : A compter de 4D v13, cette commande peut être exécutée avec 4D Server. Dans ce contexte, elle effectue en interne un appel à **QUITTER 4D** sur le serveur (entraînant l'apparition, sur chaque poste distant, d'une boîte de dialogue signalant que le serveur est en train de quitter) avant de créer le fichier désigné.

Dossier 4D

Dossier 4D {(dossier {; *})} -> Résultat

Paramètre	Type		Description
dossier	Entier long	➡	Type de dossier (si omis=dossier 4D actif)
*	Opérateur	➡	Retourner le dossier de la base hôte
Résultat	Chaîne	⮌	Chemin d'accès du dossier désigné

Description

La commande **Dossier 4D** renvoie le chemin d'accès du dossier 4D actif de l'application courante, ou du dossier de l'environnement 4D spécifié par le paramètre *dossier*, s'il est passé.
Cette commande vous permet d'obtenir avec certitude le chemin d'accès réel des dossiers utilisés par l'application. En utilisant cette commande, vous êtes certain que votre code fonctionnera correctement sur toute plate-forme, quelles que soient la langue du système et l'application 4D.

Vous pouvez passer dans *dossier* une des constantes suivantes, placées dans le thème **Environnement 4D** :

Constante	Type	Valeur
Dossier 4D actif	Entier long	0
Dossier base	Entier long	4
Dossier base 4D Client	Entier long	3
Dossier base syntaxe Unix	Entier long	5
Dossier données	Entier long	9
Dossier Licenses	Entier long	1
Dossier Logs	Entier long	7
Dossier racine HTML	Entier long	8
Dossier Resources courant	Entier long	6

Voici une description de chacun de ces dossiers :

Notes préalables sur les noms de dossiers :

- {Disque} est le disque sur lequel est installé le système.
- Le libellé Utilisateur représente le nom de l'utilisateur ayant ouvert la session.
- Avec certaines versions de Mac OS, les noms des dossiers sont traduits :
  - le dossier **Library** est nommé **Bibliothèque**,
  - le dossier **Application Support** est nommé **Support aux applications**.

Dossier 4D actif

Les applications de l'environnement 4D utilisent un dossier spécifique pour stocker les informations suivantes :

- Fichiers de préférences utilisés par les applications 4D
- Fichier shortcuts.xml (raccourcis clavier personnalisés)
- Dossier Macros v2 (macros commandes de l'éditeur de méthodes)
- Dossiers Favorites v1x, par exemple Favorites v13 (chemins d'accès des bases locales et distantes ayant été ouvertes)

Avec les applications 4D principales (4D et 4D Server), le dossier 4D actif est nommé **4D** et se trouve par défaut à l'emplacement suivant :

- Sous Windows 7 et suivants : *{Disque}:\Users\<nomUtilisateur>\AppData\Roaming\4D*
- Sous OS X : *{Disque}:Users:<nomUtilisateur>:Library:Application Support:4D*

A compter de 4D v13, dans le cas d'une application fusionnée avec 4D Volume Desktop, le dossier 4D actif se trouve à l'emplacement suivant :

- Sous Windows 7 et suivants : *{Disque}:\Users\<nomUtilisateur>\AppData\Roaming\<nomBase>*
- Sous OS X : *{Disque}:Users:<nomUtilisateur>:Library:Application Support:<nomBase>*

Dossier Licenses

Dossier contenant les fichiers de licence 4D du poste.
Le dossier **Licenses** est situé à l'emplacement suivant :

- Sous Windows 7 et suivants : *{Disque}:\ProgramData\4D\Licenses*
- Sous OS X : *{Disque}:Library:Application Support:4D:Licenses*

Notes :

- Dans le cas d'une application fusionnée avec un 4D Volume Desktop, le dossier des licences est inclus dans le package (progiciel) de l'applicatif.
- Si le dossier des licences n'a pu être créé dans le système à cause d'un défaut d'autorisation, il est créé aux emplacements suivants :
  - sous Windows 7 et suivants : *{Disque}:\Users\<nomUtilisateur>\AppData\Roaming\4D\Licenses*
  - sous OS X : *{Disque}:Users:<nomUtilisateur>:Library:Application Support:4D:Licenses*

Dossier données

Dossier contenant le fichier de données courant. Le chemin du dossier est exprimé avec la syntaxe standard de la plate-forme courante.

Dossier base 4D Client (postes clients)

Dossier de la base 4D créé en local sur chaque poste client, dans lequel sont téléchargés depuis 4D Server les dossiers et fichiers relatifs à la base (ressources, plug-ins, dossier Resources, etc.).
Le dossier **base 4D Client** est situé à l'emplacement suivant sur chaque poste client :

- Sous Windows 7 et suivants : *{Disque}:\Users\<nomUtilisateur>\AppData\Local\4D\<NomDeLaBase_Adresse>*

- Sous OS X : *{Disque}:Users:<nomUtilisateur>:Library:Caches:4D:<NomDeLaBase_Adresse>*

Dossier base

Dossier contenant le fichier de structure de la base. Le chemin d'accès est exprimé avec la syntaxe standard de la plate-forme courante.
Avec l'application 4D Client, cette constante équivaut strictement à la constante précédente Dossier base 4D Client : la commande retourne le chemin d'accès du dossier créé en local.

Dossier base syntaxe Unix

Dossier contenant le fichier de structure de la base. Cette constante désigne le même dossier que la précédente, mais le chemin d'accès retourné est exprimé avec la syntaxe Unix (Posix), du type /Users/... Cette syntaxe est principalement utile lorsque vous utilisez la commande **LANCER PROCESS EXTERNE** sous OS X.

Dossier Resources courant

Dossier Resources de la base. Ce dossier contient les éléments additionnels (images, textes) utilisés pour l'interface de la base. Un composant peut disposer de son propre dossier Resources. Le dossier Resources est situé à côté du fichier de structure de la base.
En mode client/serveur, ce dossier permet d'organiser le transfert de données personnalisées (images, fichiers, sous-dossiers...) entre le poste serveur et les postes clients. Le contenu de ce dossier est mis à jour automatiquement sur chaque client au moment de sa connexion. Tous les mécanismes de référencement associé au dossier Resources sont pris en charge en mode client/serveur (dossier .lproj, XLIFF, images...) . En outre, 4D fournit divers outils permettant de gérer et de mettre à jour dynamiquement ce dossier, notamment un Explorateur de ressources.

Note : Si le dossier **Resources** n'existe pas pour la base, l'exécution de la commande **Dossier 4D** avec la constante Dossier Resources courant provoque sa création.

Dossier Logs

Dossier Logs de la base. Ce dossier centralise les fichiers d'historique de la base courante. Il est créé au même niveau que le fichier de structure. Le dossier Logs contient les fichiers d'historique suivants :

- conversion de la base,
- requêtes du serveur Web,
- vérification et réparation des données,
- vérification et réparation de la structure,
- journal d'activités sauvegarde/restitution,
- débogage des commandes,
- requêtes 4D Server (généré sur les clients et sur le serveur)..

Note : Si le dossier **Logs** n'existe pas pour la base, l'exécution de la commande **Dossier 4D** avec la constante Dossier Logs provoque sa création.

Dossier racine HTML

Dossier racine HTML courant de la base. Le chemin d'accès retourné est exprimé avec la syntaxe standard de la plate-forme courante. Le dossier racine HTML est le dossier dans lequel le serveur Web de 4D va chercher les pages et fichiers Web demandés. Par défaut, il est nommé **DossierWeb** et est placé à côté de fichier de structure (ou de sa copie locale dans le cas de 4D en mode distant). Son emplacement peut être défini dans la page Web/Configuration des Propriétés de la base ou dynamiquement via la commande **WEB FIXER RACINE**.
Si la commande **Dossier 4D** est appelée depuis un 4D distant, le chemin retourné est celui du poste distant, pas celui de 4D Server.

Le paramètre facultatif * est utile dans le cadre d'une architecture utilisant des composants : il permet de déterminer la base (hôte ou composant) dont vous souhaitez obtenir le chemin d'accès d'un dossier. Ce paramètre est valide uniquement pour les dossiers Dossier base, Dossier base syntaxe Unix et Dossier Resources courant. Il est ignoré dans les autres cas.
Lorsque la commande est appelée depuis un composant :

- si le paramètre * est passé, la commande retourne le chemin d'accès du dossier de la base hôte,
- si le paramètre * n'est pas passé, la commande retourne le chemin d'accès du dossier du composant.
  Le dossier de la base (Dossier base et Dossier base syntaxe Unix) retourné diffère en fonction du type d'architecture du composant :
  - dans le cas d'un dossier/package .4dbase, la commande retourne le chemin d'accès du dossier/package .4dbase,
  - dans le cas d'un fichier .4db ou .4dc, la commande retourne le chemin d'accès du dossier "Components",
  - dans le cas d'un alias ou raccourci, la commande retourne le chemin d'accès du dossier contenant la base matrice originale. Le résultat diffère en fonction du format de cette base (dossier/package .4dbase ou fichier .4db/.4dc), comme décrit ci-dessus.

  Lorsque la commande est appelée depuis la base hôte, elle retourne toujours le chemin d'accès du dossier de la base hôte, que le paramètre * soit passé ou non.

Exemple 1

Pendant le démarrage d'une base mono-utilisateur, vous voulez charger (ou créer) vos propres paramètres et les stocker dans un fichier situé dans le dossier 4D. Pour cela, dans la **Méthode base Sur ouverture**, vous pouvez écrire les lignes suivantes :

```
ASSOCIER TYPES FICHIER("PREF";"PRF";"Préférences")
  ` Associer le type de fichier PREF sur Mac OS à l'extension de fichier .PRF sur Windows
$vsNomDocPref:=Dossier 4D+"MesPrefs" ` Construire le chemin d'accès au fichier Préférences
Si(Tester chemin acces($vsNomDocPref+(".PRF"*Num(Sous Windows)))#Est un document)
  ` Vérifier si le fichier existe
   $vtRefDocPref:=Creer document($vsNomDocPref;"PREF") ` Si non, il faut le créer
Sinon
   $vtRefDocPref:=Ouvrir document($vsNomDocPref;"PREF") ` Si oui, il faut l'ouvrir
Fin de si
Si(OK=1)
  ` Traiter le contenu du document
   FERMER DOCUMENT($vtRefDocPref)
Sinon
  ` Gérer l'erreur
Fin de si
```

Exemple 2

Cet exemple illustre l'emploi de la constante Dossier base syntaxe Unix sous Mac OS pour lister le contenu du dossier de la base :

```
$cheminposix:="\""+Dossier 4D(Dossier base syntaxe Unix)+"\""
$mondossier:="ls -l "+$cheminposix
$in:=""
$out:=""
$err:=""
LANCER PROCESS EXTERNE($mondossier;$in;$out;$err)
```

Note : Sous Mac OS, il est nécessaire d'encadrer les chemins d'accès avec des guillemets lorsqu'ils contiennent des noms de fichiers ou de dossiers comportant des espaces. La séquence d'échappement "\" permet d'insérer le caractère guillemets dans la chaîne. Vous pouvez également utiliser l'instruction **Caractere(Guillemets)**.

Variables et ensembles système

Si le paramètre *dossier* est invalide ou si le chemin d'accès retourné est vide, la variable système OK prend la valeur 0.

Fichier 4D

Fichier 4D (fichier {; *}) -> Résultat

Paramètre	Type		Description
fichier	Entier long	⇒	Type de fichier
*	Opérateur	⇒	Retourne le chemin d'accès du fichier de la base hôte
Résultat	Chaîne	⇐	Chemin d'accès du fichier 4D désigné

Description

La commande **Fichier 4D** retourne le chemin d'accès au fichier de l'environnement 4D spécifié par le paramètre *fichier*. Le chemin d'accès est retourné en utilisant la syntaxe système.

Cette commande vous permet de récupérer le chemin d'accès actuel à des fichiers, dont le nom et l'emplacement sont spécifiques à 4D. Elle peut aussi vous permettre d'écrire du code générique indépendant de la version de 4D et de la version de l'OS.

Dans *fichier*, passez une valeur pour désigner le fichier dont vous voulez récupérer le chemin d'accès. Vous pouvez utiliser une des constantes suivantes du thème "**Environnement 4D**" :

Constante	Type	Valeur	Comment
Fichier configuration sauvegarde	Entier long	1	Fichier Backup.xml, stocké dans le dossier Preferences/Backup à côté du fichier structure de la base.
Fichier dernière sauvegarde	Entier long	2	Dernier fichier de sauvegarde généré, nommé <NomBase>[NumBkp].4BK, stocké à un emplacement personnalisé.
Fichier propriétés structure	Entier long	3	settings.4DSettings pour tous les fichiers de données (si activé), stocké dans le dossier Preferences à côté du fichier de structure de la base
Fichier propriétés utilisateur pour données	Entier long	4	settings.4DSettings pour le fichier de données courant, stocké dans le dossier Preferences à côté du fichier de données.

Lorsque la commande est appelée à partir d'un composant, passez le paramètre optionnel * pour obtenir le chemin d'accès du *fichier* de la base hôte. Dans ce contexte, si vous omettez le paramètre *, une chaîne vide est toujours retournée.

Concernant Fichier propriétés utilisateur pour données et Fichier propriétés structure, un chemin d'accès est retourné si l'option de sécurité **Autoriser les propriétés utilisateur dans un fichier externe** est cochée dans la boîte de dialogue des Propriétés de la base (voir **Activer le mode Propriétés utilisateur**).

Exemple

Vous voulez obtenir le chemin d'accès du fichier de sauvegarde le plus récent :

```
C_TEXTE($path)
$path:=Fichier 4D(Fichier dernière sauvegarde)
 // $path = "C:\Backups\Countries\Countries[0025].4BK" par exemple
```

Fichier application

Fichier application -> Résultat

Paramètre	Type		Description
Résultat	Chaîne	➲	Nom long du fichier 4D exécutable ou de l'application 4D

Description

La fonction **Fichier application** retourne le nom long (c'est-à-dire le chemin d'accès complet au fichier, y compris son nom) du fichier exécutable ou de l'application 4D que vous utilisez.

Sous Windows
Si, par exemple, vous utilisez 4D qui se trouve dans le répertoire \PROGRAMMES\4D sur le volume E, **Fichier application** renvoie E:\PROGRAMMES\4D\4D.EXE.

Sous Mac OS
Si, par exemple, vous utilisez 4D qui se trouve dans le dossier Programmes sur le disque Disque Dur, **Fichier application** renvoie Disque Dur:Programmes:4D.app.

Exemple

Lorsque vous démarrez votre base sous Windows, vous souhaitez vérifier qu'une librairie DLL se trouve au même niveau que le fichier exécutable de 4D. Dans la **Méthode base Sur ouverture**, vous pouvez écrire les instructions suivantes :

```
Si(Sous Windows & (Type application#4D Server))
   Si(Tester chemin acces(Nom long vers chemin accès(Fichier application)+"XRAYCAPT.DLL")#Est un document)
    ` Afficher une boîte de dialogue expliquant que la librairie XRAYCAPT.DLL n'est pas présente
    ` Donc, la saisie de radios n'est pas disponible
   Fin de si
Fin de si
```

Note : Les méthodes projet ***Sous Windows*** et ***Nom long vers chemin accès*** sont détaillées dans la section **Présentation des documents système**.

Fichier donnees

Fichier donnees {(segment)} -> Résultat

Paramètre	Type		Description
segment	Entier long	⇒	Obsolète, ne pas utiliser
Résultat	Chaîne	⇐	Nom long du fichier de données de la base

Description

La fonction **Fichier donnees** retourne le nom long (c'est-à-dire le chemin d'accès complet au fichier, y compris son nom) du fichier de données de la base avec laquelle vous êtes en train de travailler.

Depuis la version 11 de 4D, les segments de données ne sont plus pris en charge. Le paramètre *segment* est désormais ignoré, il ne doit plus être utilisé.

Sous Windows
Si, par exemple, vous travaillez avec la base MesCDs qui se trouve à l'emplacement \DOCS\MesCDs sur le volume G, **Fichier donnees** retournera G:\DOCS\MesCDs\MesCDs.4DD (si vous avez choisi l'emplacement proposé par défaut par 4D lorsque vous avez créé la base).

Sous Mac OS
Si, par exemple, vous travaillez avec la base MesCDs qui se trouve dans le dossier Documents:MesCDs*ƒ*: sur le disque Macintosh HD, **Fichier donnees** retournera Macintosh HD:Documents:MesCDs*ƒ*:MesCDs.data (si vous avez choisi l'emplacement proposé par défaut par 4D lorsque vous avez créé la base).

ATTENTION : Si vous appelez cette fonction depuis 4D en mode distant, seul le nom du fichier de données est retourné, pas le nom long.

Fichier donnees verrouille

Fichier donnees verrouille -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai = fichier/segment verrouillé Faux = fichier/segment non verrouillé

Description

La commande **Fichier donnees verrouille** retourne Vrai si le fichier de données de la base ouverte ou l'un de ses segments au moins est verrouillé — c'est-à-dire, protégé en écriture.

Placée par exemple dans la **Méthode base Sur ouverture**, cette commande permet de prévenir tout risque d'ouverture fortuite d'un fichier de données verrouillé.

Exemple

Cette méthode empêchera l'ouverture de la base si le fichier de données est verrouillé :

```
Si(Fichier donnees verrouille)
   ALERTE("Le fichier de données est verrouillé. Impossible d'ouvrir la base.")
   QUITTER 4D
Fin de si
```

Fichier structure

Fichier structure {(*)} -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Retourner le fichier de structure de la base hôte
Résultat	Chaîne	➲	Nom long du fichier de structure de la base

Description

La fonction **Fichier structure** retourne le nom long (c'est-à-dire le chemin d'accès complet au fichier, y compris son nom) du fichier de structure de la base en cours d'utilisation.

Sous Windows
Si, par exemple, vous travaillez sur la base MesCDs qui se trouve à \DOCS\MesCDs sur le volume G, **Fichier structure** renvoie G:\DOCS\MyCDs\MesCDs.4DB.

Sous Macintosh
Si, par exemple, vous travaillez sur la base MesCDs qui se trouve dans le dossier Documents:MesCDs*ƒ*: sur le disque qui s'appelle Macintosh HD, **Fichier structure** renvoie Macintosh HD:Documents:MesCDs*ƒ*:MesCDs.

Note : Dans le cas particulier d'une base compilée et fusionnée avec 4D Volume Desktop, cette commande retourne le chemin d'accès du fichier de l'application (fichier exécutable) sous Windows et OS X. Sous OS X, ce fichier est situé à l'intérieur du progiciel, dans le dossier [Contents:MacOS]. Ce fonctionnement provient d'un ancien mécanisme, conservé pour des raisons de compatibilité. Si vous souhaitez obtenir le nom long du progiciel lui-même, il est préférable d'utiliser la commande **Fichier application**. L'astuce consiste à tester l'application à l'aide de la commande **Type application** puis à exécuter **Fichier structure** ou **Fichier application** en fonction du contexte.

ATTENTION : Si vous appelez cette commande lorsque vous utilisez 4D en mode distant, seul le nom du fichier de structure est renvoyé, pas le nom long.

Le paramètre facultatif * est utile dans le cadre d'une architecture utilisant des composants : il permet de déterminer la structure (hôte ou composant) dont vous souhaitez obtenir le nom long en fonction du contexte d'appel de la commande :

- Lorsque la commande est appelée depuis un composant :
  - si le paramètre * est passé, la commande retourne le nom long du fichier de structure de la base hôte
  - si le paramètre * n'est pas passé, la commande retourne le nom long du fichier de structure du composant.
    Le fichier de structure d'un composant correspond au fichier .4db ou .4dc du composant situé dans le dossier "Components" de la base.
    Cependant, un composant peut également être installé sous la forme d'un alias/raccourci ou d'un dossier/package .4dbase :
    - dans le cas d'un composant installé sous forme d'alias/raccourci, la commande retourne le chemin d'accès du fichier .4db ou .4dc original (l'alias ou le raccourci est résolu).
    - dans le cas d'un composant installé sous forme de dossier/package .4dbase, la commande retourne le chemin d'accès du fichier .4db ou .4dc à l'intérieur de ce dossier/package.
- Lorsque la commande est appelée depuis une méthode d'une base hôte, elle retourne toujours le nom long du fichier de structure de la base hôte, que le paramètre * soit passé ou non.

Exemple 1

Cet exemple affiche le nom et l'emplacement du fichier de structure que vous utilisez :

```
Si(Type application#4D mode distant)
   $vsStructNomFichier:=Nom long vers nom de fichier(Fichier structure)
   $vsStructNomChemin:=Nom long vers chemin accès(Fichier structure)
   ALERTE("Vous êtes en train d'utiliser la base "+Caractere(34)+$vsStructNomFichier+Caractere(34)+" qui se trouve
au "+Caractere(34)+$vsStructNomChemin+Caractere(34)+".")
Sinon
   ALERTE("Vous êtes connecté à la base "+Caractere(34)+Fichier structure+Caractere(34))
Fin de si
```

Note : Les méthodes projet ***Nom long vers nom de fichier*** et ***Nom long vers chemin accès*** sont détaillées dans la section **Présentation des documents système**.

Exemple 2

L'exemple suivant permet de savoir si la méthode est appelée depuis un composant :

```
C_BOOLEEN($0)
$0:=(Fichier structure#Fichier structure(*))
  ` $0=Vrai si la méthode est appelée depuis un composant
```

FIXER DOSSIER MISE A JOUR

FIXER DOSSIER MISE A JOUR (cheminDossier {; erreursDiscrètes})

Paramètre	Type		Description
cheminDossier	Chaîne	➡	Chemin d'accès du dossier (package sous OS X) contenant l'application mise à jour
erreursDiscrètes	Booléen	➡	Faux (défaut) = afficher des messages d'erreur, Vrai = ne pas afficher de messages (uniquement enregistrer les erreurs)

Description

La commande **FIXER DOSSIER MISE A JOUR** permet de définir le dossier contenant la mise à jour de l'application 4D fusionnée courante. Cette information est mémorisée durant la session 4D jusqu'à l'appel de la commande **REDEMARRER 4D**. Si l'application est quittée manuellement, cette information n'est pas conservée.

Cette commande est destinée à être utilisée dans un processus de mise à jour automatique d'une application fusionnée (serveur ou monoposte). Pour plus d'informations, reportez-vous à la section **Finaliser et déployer les applications finales** dans le manuel *Mode Développement*.

Note : La commande fonctionne uniquement avec 4D Server ou une application monoposte fusionnée avec 4D Volume Desktop.

Passez dans le paramètre *cheminDossier* le chemin d'accès complet du dossier de la nouvelle version de l'application fusionnée (dossier contenant l'application *monApp4D.exe* sous Windows et package *monApp4D.app* sous OS X), créée par le générateur d'applications de 4D.

Note : Il est fortement conseillé d'utiliser pour les fichiers des nouvelles versions des applications le même nom que ceux des applications elles-mêmes, car le processus de mise à jour remplace le dossier de l'application. Si vous utilisez des noms différents, les raccourcis et chemins mémorisés ne fonctionneront plus.

Si les paramètres sont valides, la mise à jour est placée "en attente" dans la session jusqu'à l'appel de la commande **REDEMARRER 4D**. Si vous exécutez plusieurs fois **FIXER DOSSIER MISE A JOUR** avant **REDEMARRER 4D**, le dernier appel valide est pris en compte.

Vous pouvez passer une chaîne vide ("") dans *cheminDossier* pour réinitialiser les informations de mise à jour pour la session courante.

Le paramètre optionnel *erreursDiscrètes* permet de définir le mode de report des erreurs lors de la mise à jour :

- si vous passez **Faux** ou si ce paramètre est omis, les erreurs sont inscrites dans le journal des mises à jour et affichées dans une boîte de dialogue d'alerte.
- si vous passez **Vrai**, les erreurs sont uniquement inscrites dans le journal des mises à jour.

Exception : s'il n'est pas possible de créer un fichier journal, une boîte de dialogue d'alerte est affichée, quelle que soit la valeur du paramètre *erreursDiscrètes*. Pour plus d'informations, reportez-vous à la description de la commande **Lire chemin journal derniere mise a jour**.

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0. Vous pouvez intercepter les erreurs éventuellement générées par la commande à l'aide d'une méthode installée via la commande **APPELER SUR ERREUR**.

Exemple

Vous avez créé un dossier "MesMisesAJour" sur votre disque, dans lequel vous avez placé une nouvelle version de l'application "MonAppli". Vous ne souhaitez pas afficher les erreurs. Pour préparer la mise à jour, vous écrivez :

```
    // Syntaxe Windows
FIXER DOSSIER MISE A JOUR("C:\\MesMisesAJour"+Séparateur dossier+"MonAppli"+Séparateur dossier;Vrai)

    // Syntaxe OS X
FIXER DOSSIER MISE A JOUR("MacHD:MesMisesAJour"+Séparateur dossier+"MonAppli.app"+Séparateur dossier;Vrai)
```

FIXER LANGUE BASE

FIXER LANGUE BASE (codeLangue {; *})

Paramètre	Type		Description
codeLangue	Texte	⇒	Sélecteur de langue
*	Opérateur	⇒	Portée de la commande

Description

La commande **FIXER LANGUE BASE** permet de modifier la langue courante de la base pour la session courante.

La langue courante de la base permet de définir le dossier .lproj dans lequel le programme va chercher les éléments localisés de l'application (textes et images). Par défaut, 4D détermine automatiquement la langue courante en fonction du contenu du dossier **Resources** et de l'environnement système (cf. description de la commande **Lire langue base**). **FIXER LANGUE BASE** vous permet de modifier la langue courante par défaut.

La commande ne modifie pas la langue des formulaires déjà chargés, seuls les éléments affichés postérieurement à l'appel de la commande tiendront compte du nouveau paramétrage.

Passez dans *codeLangue* la langue à utiliser pour l'application, exprimée dans la norme définie par la RFC 3066, ISO639 et ISO3166. Typiquement, vous devez passer "fr" pour le français, "es" pour l'espagnol, "en-us" pour l'anglais américain, etc. Pour plus d'informations sur cette norme, reportez-vous à l'**MissingRef** dans le manuel *Mode Développement*.

Par défaut, la commande s'applique à toutes les bases et composants ouverts, pour tous les process. Le paramètre optionnel *, s'il est passé, spécifie que la commande doit s'appliquer uniquement à la base qui a l'a appelée. Ce paramètre permet en particulier de définir séparément la langue de la base et celle d'un composant.

Si la commande est correctement exécutée, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Note : Conformément à la RFC, la commande utilise le "-" (tiret) pour séparer le code langue et le code région, par exemple "fr-ca" ou "en-us". En revanche, les dossiers "lproj" du dossier **Resources** utilisent le "_" (soulignement), par exemple "fr_ca.lproj" ou "en_us.lproj".

4D Server : Avec 4D Server, les langues disponibles sont celles situées sur le poste distant ayant appelé la commande. Vous devez donc veiller à la synchronisation des dossiers **Resources**.

Exemple 1

Nous souhaitons définir la langue de l'interface en français :

```
FIXER LANGUE BASE("fr")
```

Exemple 2

L'interface de votre application utilise la chaîne statique ":xliff:shopping". Les fichiers xliff contiennent notamment les informations suivantes :

- Dossier FR :

```
<trans-unit id="15" resname="Shopping">   <source>Shopping</source>   <target>Faire les courses</target>
</trans-unit>
```

- Dossier FR_CA :

```
<trans-unit id="15" resname="Shopping">   <source>Shopping</source>   <target>Magasiner</target> </trans-
unit>
```

```
FIXER LANGUE BASE("fr")
 //La chaîne ":xliff:shopping" affiche "Faire les courses"
FIXER LANGUE BASE("fr-ca")
 //La chaîne ":xliff:shopping" affiche "Magasiner"
```

FIXER PARAMETRE BASE

FIXER PARAMETRE BASE ({laTable ;} sélecteur ; valeur)

Paramètre	Type		Description
laTable	Table	⇒	Table à paramétrer ou Table par défaut si ce paramètre est omis
sélecteur	Entier long	⇒	Code du paramètre de la base à modifier
valeur	Réel, Chaîne	⇒	Valeur du paramètre

Description

La commande **FIXER PARAMETRE BASE** permet de modifier divers paramètres internes de la base de données 4D.

sélecteur désigne le paramètre à modifier. 4D vous propose des constantes prédéfinies, placées dans le thème **Paramètres de la base**. Le tableau suivant décrit chaque constante et indique sa portée et sa persistance entre deux sessions :

Constante	Type	Valeur	Comment
_o_Enreg requêtes Web	Entier long	29	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
_o_Mode conversion Web	Entier long	8	***** Sélecteur inactivé *****
_o_Précision affichage réels	Entier long	32	***** Sélecteur inactivé *****
_o_Taille cache données	Entier long	9	**Portée** : Application 4D **Conservé entre deux sessions** : - **Description** : *Constante obsolète (conservée uniquement pour des raisons de compatibilité).* L'utilisation de la commande **Lire taille cache** est désormais recommandée.
Adresse IP d'écoute	Entier long	16	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Appels système 4D mode distant	Entier long	12	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Valeurs possibles** : pour les sélecteurs 10, 11 et 12, le paramètre *valeur* est de la forme hexadécimale *0x00aabbcc* dans laquelle : *aa* = nombre minimum de ticks par appel au système (de 0 à 100 inclus) *bb* = nombre maximum de ticks par appel au système (de 0 à 100 inclus) *cc* = nombre de ticks entre deux appels au système (de 0 à 20 inclus). Si une des valeurs est hors de son intervalle, 4D la fixe à son maximum. Vous pouvez également passer dans *valeur* une des trois valeurs suivantes, correspondant à un ensemble de réglages standard : • *valeur* = -1 : priorité maximum allouée à 4D, • *valeur* = -2 : priorité moyenne allouée à 4D, • *valeur* = -3 : priorité minimum allouée à 4D. **Description** : Ce sélecteur vous permet de fixer dynamiquement les réglages des appels système de 4D. En fonction du Sélecteur, le gestionnaire d'appels système sera défini pour : • 4D mode local lorsque la commande est appelée depuis 4D monoposte (*sélecteur=10*). • 4D Server lorsque la commande est appelée depuis 4D Server (*sélecteur=11*). • 4D mode distant lorsque la commande est appelée depuis 4D connecté à 4D Server (*sélecteur=12*). **Note :** Le fonctionnement du sélecteur 12 (Appels système 4D mode distant) diffère suivant que la commande **FIXER PARAMETRE BASE** est exécutée sur le poste serveur ou sur un poste client : - si la commande est exécutée sur le poste serveur, la nouvelle valeur sera appliquée à tous les postes clients se connectant ultérieurement. - si la commande est exécutée sur un poste client, la nouvelle valeur est appliquée immédiatement au poste client ainsi qu'à tous les postes clients se connectant par la suite au serveur. Vous pouvez utiliser ce fonctionnement pour mettre en place une gestion dynamique et individualisée de la priorité pour chaque poste client. Le principe consiste à exécuter la commande une première fois sur le poste client à configurer puis une seconde fois sur le poste serveur avec la valeur par défaut, qui sera alors utilisée pour les postes client se connectant par la suite. Ce fonctionnement est effectif dans 4D à partir des versions 6.8.6, 2003.3 et 2004. **Attention** : Paramétrer ces sélecteurs de façon inappropriée peut conduire à une forte dégradation des performances de l'application. Il est conseillé de ne modifier les valeurs par défaut qu'en parfaite connaissance de cause.
Appels système 4D mode local	Entier long	10	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Description** : voir sélecteur 12
Appels système 4D Server	Entier long	11	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Description** : voir sélecteur 12
			Portée : Base de données

Constante	Type	Valeur	Commentaire
Casse caractères moteur SQL	Entier long	44	**Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (casse non prise en compte) ou 1 (casse prise en compte) **Description** : Activation ou désactivation de la prise en compte de la casse des caractères pour les comparaisons de chaînes effectuées par le moteur SQL. Par défaut, la valeur est 1 (casse prise en compte) : le moteur SQL établit une différence entre les majuscules et les minuscules ainsi qu'entre les caractères accentués lors des comparaisons de chaînes (tris et recherches). Par exemple "ABC" = "ABC" mais "ABC" # "Abc" et "abc" # "âbc" . Dans certains cas, par exemple pour aligner le fonctionnement du moteur SQL sur celui du moteur 4D, vous pourrez souhaiter que les comparaisons de chaînes ne tiennent pas compte de la casse ("ABC"="Abc"="âbc"). Cette option peut également être définie dans la **Page SQL** des Propriétés de la base.
Chercher par formule serveur	Entier long	46	**Portée** : Table et process courants **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (utiliser le paramétrage de la base), 1 (exécuter sur le client) ou 2 (exécuter sur le serveur) **Description** : Emplacement de l'exécution des commandes **CHERCHER PAR FORMULE** et **CHERCHER PAR FORMULE DANS SELECTION** pour la *table* passée en paramètre. Dans le cadre de l'exploitation d'une base en client-serveur, les commandes de recherche "par formule" peuvent exécutées soit sur le serveur soit sur le client : • dans les bases de données créées à partir de 4D v11 SQL, ces commandes sont exécutées sur le serveur. • dans les bases de données converties, ces commandes sont exécutées sur le client, comme dans les versions précédentes de 4D. • dans les bases de données converties, une préférence spécifique permet de modifier globalement le lieu d'exécution de ces commandes. Cette différence de lieu d'exécution influe sur les performances de l'application (l'exécution sur le serveur est généralement plus rapide) mais également sur la programmation. En effet, la valeur des composantes de la formule (notamment les variables appelées via une méthode) diffère suivant le contexte d'exécution. Vous pouvez utiliser ce sélecteur pour adapter ponctuellement le fonctionnement de votre application. Si vous passez 0 dans le paramètre *valeur*, l'emplacement d'exécution des commandes de recherche "par formule" dépendra de la configuration de la base : dans les bases créées avec 4D v11 SQL, les commandes seront exécutées sur le serveur. Dans les bases converties, elles seront exécutées sur le client ou le serveur en fonction des préférences de la base. Passez 1 ou 2 dans *valeur* pour "forcer" l'exécution des commandes respectivement sur le client ou sur le serveur. Reportez-vous à l'exemple 4. **Note :** Si vous souhaitez pouvoir activer les jointures "type SQL" (cf. sélecteur Jointures chercher par formule), vous devez toujours exécuter les formules sur le serveur afin qu'elle ait accès aux enregistrements. Attention, dans ce contexte, la formule ne doit pas contenir d'appel à une méthode, sinon elle est automatiquement basculée sur le poste distant.
Client adresse IP d'écoute	Entier long	23	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 16 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Propriétés de la base 4D en mode distant.
Client enreg requêtes Web	Entier long	30	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 = Ne pas enregistrer (défaut), 1 = Enregistrer au format CLF, 2 = Enregistrer au format DLF, 3 = Enregistrer au format ELF, 4 = Enregistrer au format WLF. **Description** : Démarrage ou arrêt de l'enregistrement des requêtes Web reçues par les serveurs Web de tous les postes clients. Par défaut, la valeur est 0 (pas d'enregistrement des requêtes). Le fonctionnement de ce sélecteur est identique à celui du sélecteur 29 ; il s'applique toutefois à tous les postes 4D clients utilisés en tant que serveurs Web. Le fichier "logweb.txt" est dans ce cas automatiquement placé dans le sous-dossier Logs du dossier base 4D client (dossier de cache). Si vous souhaitez définir des valeurs pour certains postes clients uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client jeu de caractères	Entier long	24	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 17 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client maximum process Web	Entier long	20	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 7 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client minimum process Web	Entier long	19	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 6 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client numéro de port	Entier long	22	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 15 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des

Constante	Type	Valeur	Commentaire
			Préférences de 4D en mode distant. **Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 à 65535
Client numéro de port HTTPS	Entier long	40	**Description** : Numéro du port TCP utilisé par les serveurs Web des postes clients pour les connexions sécurisées via SSL (protocole HTTPS). Par défaut, la valeur est 443 (valeur standard). Le fonctionnement de ce sélecteur est identique à celui du sélecteur 39 ; il s'applique toutefois à tous les postes 4D distants utilisés en tant que serveurs Web. Si vous souhaitez modifier la valeur de certains postes clients uniquement, utilisez la boîte de dialogue des Préférences de 4D distant.
Client proc Web simultanés maxi	Entier long	25	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 18 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client taille max requêtes Web	Entier long	21	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 27 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Correcteur orthographique	Entier long	81	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (défaut) = correcteur OS X natif (Hunspell désactivé), 1 = correcteur Hunspell actif. **Description** : Permet d'activer le correcteur orthographique Hunspell sous OS X. Par défaut, sur cette plate-forme le correcteur natif est activé. Vous pouvez souhaiter utiliser le correcteur Hunspell par exemple pour unifier l'interface de vos applications multiplates-formes (sous Windows, seul le correcteur Hunspell est disponible). Pour plus d'informations, reportez-vous au paragraphe **Prise en charge des dictionnaires Hunspell**.
Direct2D lire statut actif	Entier long	74	**Note :** Ce sélecteur peut être utilisé uniquement avec la commande **Lire parametre base**, sa valeur ne peut pas être fixée. **Description** : Retourne l'implémentation active de Direct2D sous Windows. **Valeurs possibles** : 0, 1, 2, 3, 4 ou 5 (cf. valeurs du sélecteur 69). La valeur retournée dépend de la disponibilité de Direct2D, du matériel et de la qualité de la prise en charge de Direct2D par le système d'exploitation. Par exemple, si vous exécutez : `FIXER PARAMETRE BASE(Direct2D Statut;Direct2D Matériel)` `$mode:=Lire parametre base(Direct2D Lire statut actif)` - sur Windows 7 et suivants, *$mode* vaudra 1 si le système détecte un matériel compatible Direct2D, sinon *$mode* vaudra 3 (contexte logiciel). - sur Windows Vista, *$mode* vaudra 1 si le système détecte un matériel compatible Direct2D, sinon *$mode* vaudra 0 (désactivation de Direct2D). - sur Windows XP, *$mode* vaudra toujours 0 (incompatibilité avec Direct2D).
Direct2D Logiciel	Entier long	3	Voir sélecteur 69 (Direct2D Statut)
Direct2D statut	Entier long	69	**Portée**: Application 4D **Conservé entre deux sessions** : Non **Description** : Mode d'activation de l'implémentation de Direct2D sous Windows. **Valeurs possibles** : Une des constantes suivantes (mode 3 par défaut) : Direct2D désactivé (0) : le mode Direct2D n'est pas activé, la base fonctionne dans le mode précédent (GDI/GDIPlus). Direct2D matériel (1) : utilisation de Direct2D en contexte graphique matériel dans toute l'application 4D. Si ce contexte n'est pas disponible, utilisation du contexte graphique Direct2D logiciel (hormis sous Vista, pour des raisons de performances dans ce cas le mode GDI/GDIPlus est utilisé). Direct2D Logiciel (3) (Mode par défaut) : à partir de Windows 7, utilisation de Direct2D en contexte graphique logiciel dans toute l'application 4D. Sous Vista, pour des raisons de performances le mode GDI/GDIPlus est utilisé. **Note de compatibilité :** A compter de 4D v14, les modes hybrides sont désactivés et sont redirigés vers les modes disponibles (l'ancien mode 2 équivaut au mode 1 ; les anciens modes 4 et 5 sont équivalents au mode 3).
Enreg diagnostic	Entier long	79	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 ou 1 (0 = ne pas enregistrer, 1 = enregistrer) **Description** : Démarrage ou arrêt de l'enregistrement du fichier de diagnostic de 4D. Par défaut, la valeur est 0 (pas d'enregistrement). 4D vous permet d'enregistrer de manière continue dans un fichier de diagnostic un ensemble d'événements relatifs au fonctionnement interne de l'application. Les informations contenues dans ce fichier sont destinées à la mise au point des applications 4D et pourront être analysées avec l'aide des services techniques de 4D. Lorsque vous passez 1 dans ce sélecteur, le fichier de diagnostic, nommé *NomBase*.txt, est automatiquement créé (ou ouvert) dans le dossier **Logs** de la base. Une fois que le fichier atteint une taille de 10 Mo, il est refermé et un nouveau fichier *NomBase*_N.txt est généré, avec un numéro séquentiel N incrémenté. A noter qu'il est possible d'inclure des informations personnalisées dans ce fichier à l'aide de la commande **ENREGISTRER EVENEMENT**.
			Portée : Application 4D **Conservé entre deux sessions** : Non **Description** : Démarrage ou arrêt de l'enregistrement séquentiel des événements de programmation de 4D dans le fichier *4DDebugLog*, automatiquement placé dans le sous-dossier Logs de la base, à côté du fichier de structure. Un nouveau format texte tabulé, plus compact, est utilisable pour le fichier d'enregistrement des événements "4DDebugLog[_n].txt" à compter de 4D v14 (_n est le numéro de segment du fichier). **Valeurs possibles** : Entier long contenant un champ de bits (bit field) : valeur = bit1(1)+bit2(2)+bit3(4)+bit4(8)+…).

Constante	Type	Valeur	Comment
Enreg événements debogage	Entier long	34	Le bit 1 (valeur 1) permet de demander à activer le fichier (à noter que toute autre valeur non nulle l'activera également) - Le bit 2 (valeur 2) permet de demander les paramètres d'appel aux méthodes et commandes. - Le bit 3 (valeur 4) permet d'activer le nouveau format tabulé. - Le bit 4 (valeur 8) permet de désactiver l'écriture immédiate de chaque opération sur disque (activée par défaut). L'écriture immédiate est moins rapide mais plus efficace par exemple pour rechercher les causes d'un plantage. Si vous désactivez ce mode, le contenu fichier sera plus compact et il sera généré plus rapidement. - Le bit 5 (valeur 16) permet de désactiver l'enregistrement des appels de plug-ins (activé par défaut). Dans le format non tabulé (ancien format), les durées d'exécution sont exprimées en millisecondes, la valeur "< ms" est affichée si une opération s'exécute en moins d'une milliseconde. Dans le nouveau format tabulé, les durées d'exécution sont exprimées en microsecondes. Exemples : FIXER PARAMETRE BASE(34;1) // active le fichier mode v13 sans les paramètres, avec les durées FIXER PARAMETRE BASE(34;2) // active le fichier mode v13 avec les paramètres et les durées FIXER PARAMETRE BASE(34;2+4) // active le fichier au format v14 avec les paramètres et les durées FIXER PARAMETRE BASE(34;0) // désactive le fichier Afin d'éviter que le fichier n'enregistre une trop grande quantité d'informations, vous pouvez restreindre les commandes 4D à examiner à l'aide du sélecteur 80, Liste commandes enreg. L'option peut être activée dans tout type d'application 4D (4D tous modes, 4D Server, 4D Volume Desktop), en interprété ou en compilé. **Note :** Cette option est proposée uniquement à des fins de débogage, elle ne doit pas être utilisée en production car elle peut entraîner une dégradation des performances de l'application ainsi que la saturation du disque dur. Pour plus d'informations sur le format et l'exploitation du fichier 4DDebugLog[_n].txt, veuillez contacter les services techniques de 4D SAS. **Portée** : *4D Server, 4D distant* **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 ou de 1 à N (0 = ne pas enregistrer, 1 à N = numéro séquentiel, accolé au nom du fichier).
Enreg requêtes 4D Server	Entier long	28	**Description** : Démarrage ou arrêt de l'enregistrement des requêtes standard reçues par 4D Server (hors requêtes Web). Par défaut, la valeur est 0 (pas d'enregistrement de requêtes). 4D Server vous permet d'enregistrer dans un fichier d'historique chaque requête reçue par le poste serveur. Lorsque ce mécanisme est activé, deux fichiers sont créés dans le dossier Logs de la base, à côté du fichier de structure. Il sont nommés 4DRequestsLog_N.txt et 4DRequestsLog_ProcessInfo_N.txt où N est le numéro séquentiel de l'historique. Une fois qu'un fichier atteint une taille de 10 Mo, il est refermé et un nouveau fichier est généré, avec un numéro séquentiel incrémenté. Si un fichier du même nom existe déjà, il est directement remplacé. Vous pouvez définir le numéro de départ de la séquence à l'aide du paramètre *valeur*. Ces fichiers texte stockent dans un format tabulé simple diverses informations concernant chaque requête : heure, numéro de process, utilisateur, taille de la requête, durée de traitement, etc. Ces informations sont particulièrement utiles en phase de mise au point de l'application ou à des fins statistiques. Elles peuvent être importées par exemple dans un logiciel tableur afin d'être traitées.
Enreg requêtes client	Entier long	45	**Portée** : Poste 4D distant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 ou de 1 à N (0 = ne pas enregistrer, 1 à N = numéro séquentiel, accolé au nom du fichier). **Description** : Démarrage ou arrêt de l'enregistrement des requêtes standard effectuées par le poste client 4D ayant exécuté la commande (hors requêtes Web). Par défaut, la valeur est 0 (pas d'enregistrement des requêtes). 4D vous permet d'enregistrer l'historique des requêtes effectuées par le poste client. Lorsque ce mécanisme est activé, deux fichiers sont créés sur le poste client, dans le sous-dossier Logs du dossier local de la base. Il sont nommés 4DRequestsLog_N.txt et 4DRequestsLog_ProcessInfo_N.txt où N est le numéro séquentiel de l'historique. Une fois que le fichier 4DRequestsLog atteint une taille de 10 Mo, il est refermé et un nouveau fichier est généré, avec un numéro séquentiel incrémenté. Si un fichier du même nom existe déjà, il est directement remplacé. Vous pouvez définir le numéro de départ de la séquence à l'aide du paramètre valeur. Ces fichiers texte stockent dans un format tabulé simple diverses informations concernant chaque requête : heure, numéro de process, taille de la requête, durée de traitement, etc. Ces informations sont particulièrement utiles en phase de mise au point de l'application ou à des fins statistiques.
Inversion des objets	Entier long	37	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0, 1 ou 2 (0 = mode désactivé, 1 = mode automatique, 2 = mode activé). **Description** : Configuration du mode "inversion des objets". Ce mode permet d'inverser en mode Application les formulaires, objets, menus, etc. lorsque la base est affichée sous Windows dans une langue de droite à gauche. Ce mode peut également être configuré dans la page Interface/langue droite à gauche des Propriétés de la base. • La valeur 0 indique que le mode n'est jamais activé, quelle que soit la configuration du système (correspond à la valeur Jamais dans les Propriétés de la base). • La valeur 1 indique que le mode est activé ou non en fonction de la configuration du système (correspond à la valeur Automatique dans les Propriétés de la base). • La valeur 2 indique que le mode est activé, quelle que soit la configuration du système (correspond à la valeur Toujours dans les Propriétés de la base). Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.
Jeu de caractères	Entier long	17	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
			Portée : Process courant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (utiliser paramétrages de la base), 1 (toujours utiliser les liens auto) ou 2 (utiliser les jointures SQL si possible). **Description** : Mode de fonctionnement des commandes **CHERCHER PAR FORMULE** et **CHERCHER PAR FORMULE DANS SELECTION** relatif à l'utilisation de "jointures SQL". Dans les bases de données créées à compter de la version 11.2 de 4D v11 SQL, ces commandes effectuent des jointures sur le modèle des jointures SQL. Ce mécanisme permet de modifier la sélection d'une table en fonction d'une recherche effectuée sur une autre table sans que les tables soient reliées par un lien automatique (condition nécessaire dans les versions précédentes de 4D).

Constante	Type	Valeur	Comment
Jointures chercher par formule	Entier long	49	Le sélecteur Jointures chercher par formule vous permet de définir le mode de fonctionnement des commandes de recherche par formule pour le process courant : • 0 : utiliser les paramètres courants de la base (valeur par défaut). Dans les bases de données créées à compter de la version 11.2 de 4D v11 SQL, les "jointures SQL" sont toujours activées pour les recherches par formule. Dans les bases de données converties, ce mécanisme est inactivé par défaut pour des raisons des compatibilité mais peut être mis en oeuvre via une préférence. • 1 : toujours utiliser les liens auto (= fonctionnement des versions précédentes de 4D). Dans ce mode, un lien est nécessaire pour définir la sélection d'une table en fonction de recherches effectuées dans une autre table. 4D n'effectue pas de "jointures SQL". • 2 : utiliser les jointures SQL si possible (= fonctionnement par défaut des bases créées en version 11.2 et suivantes de 4D v11 SQL). Dans ce mode, 4D établit des "jointures SQL" pour les recherches par formule lorsque la formule s'y prête (à deux exceptions près, voir la description de la commande commandes **CHERCHER PAR FORMULE** ou **CHERCHER PAR FORMULE DANS SELECTION**. **Note :** Avec 4D en mode distant, les "jointures SQL" ne peuvent être utilisées que si les formules sont exécutées sur le serveur (elles doivent avoir accès aux enregistrements). Pour configurer le lieu d'exécution des formules, reportez-vous aux sélecteurs 46 et 47.
JSON fuseau horaire local	Entier long	85	**Portée** : Process courant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 = ignorer le fuseau horaire local, 1 (défaut) = tenir compte du fuseau horaire. **Description** : Par défaut, la conversion des dates 4D vers le format JSON tient compte du fuseau horaire local. Par exemple, la conversion de la date !23/08/2013! donne "2013-08-22T22:00:00Z" au format JSON lorsque l'opération est effectuée en France en été (GMT+2). Ce principe est conforme au fonctionnement standard de JavaScript. Ce fonctionnement peut être source d'erreurs si vous souhaitez envoyer des valeurs de date en JSON à une personne qui se trouve dans un autre fuseau horaire. C'est le cas par exemple pour l'exportation d'une table avec **Selection vers JSON** en France destiné à être réimporté aux USA avec **JSON VERS SELECTION**. Par défaut, les dates étant réinterprétées dans chaque fuseau horaire, les valeurs stockées dans la base seront différentes. Dans ce cas, vous pouvez modifier le mode de conversion des dates afin qu'il ne tienne pas compte du fuseau horaire en passant 0 dans ce sélecteur. La conversion de la date !23/08/2013! donnera alors "2013-08-23T00:00:00Z" dans tous les cas.
Limitation nombre journaux	Entier long	90	**Portée** : 4D local, 4D Server. **Conservé entre deux sessions** : Non **Valeurs possibles** : Toute valeur entière, 0 = conserver tous les journaux **Description** : Nombre maximum de fichiers à conserver par roulement pour chaque type de journal. Par défaut, tous les fichiers sont conservés. Si vous passez une valeur N, seuls les N fichiers les plus récents seront conservés, le plus ancien étant automatiquement effacé à la création d'un nouveau. Ce paramétrage s'applique à chacun des fichiers journaux suivants : journal des requêtes (sélecteurs 28 et 45), journal de débogage (sélecteur 34), journal des événements (sélecteur 79), ainsi que l'historique des requêtes Web et l'historique debug des requêtes Web (sélecteurs 29 et 84 de la commande **WEB FIXER OPTION**).
Liste commandes enreg	Chaîne	80	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Chaîne contenant la liste des numéros des commandes 4D à enregistrer (séparées par des points-virgules), "all" pour enregistrer toutes les commandes ou "" (chaîne vide) pour n'enregistrer aucune commande. **Description** : Liste des commandes 4D à enregistrer dans le fichier de débogage (cf. sélecteur 34, Enreg événements debogage). Par défaut, toutes les commandes 4D sont enregistrées. Ce sélecteur vous permet de restreindre la quantité d'informations stockées dans le fichier de débogage en limitant les commandes 4D dont vous souhaitez enregistrer l'exécution. Par exemple, vous pouvez écrire : `FIXER PARAMETRE BASE(Liste commandes enreg;"277;341") //enregistrer uniquement les commandes CHERCHER et CHERCHER DANS SELECTION`
Liste de chiffrement SSL	Chaîne	64	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Suite de chaînes séparées par des deux-points **Description :** Liste de chiffrement (*cipher list*) utilisée par 4D pour le protocole sécurisé. Cette liste permet de modifier la priorité des algorithmes de chiffrement mis en oeuvre par 4D. Par exemple, vous pouvez passer la chaîne suivante dans le paramètre *valeur* : "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". Pour une description complète de la syntaxe de la liste de chiffrement, reportez-vous à la page ciphers sur le site de OpenSSL. Ce paramétrage est global à l'application 4D (il concerne le serveur HTTP, le serveur SQL, les connexions client/serveur ainsi que le client HTTP et les commandes 4D faisant appel au protocole sécurisé). Lorsque la liste de chiffrement a été modifiée, vous devez redémarrer le serveur concerné pour que le nouveau paramétrage soit pris en compte. Pour réinitialiser la liste de chiffrement à sa valeur par défaut (stockée en dur dans le fichier SLI), appelez la commande **FIXER PARAMETRE BASE** et passez une chaîne vide ("") dans le paramètre *valeur*. **Note :** Avec la commande **Lire parametre base**, la liste de chiffrement est retournée dans le paramètre optionnel *valeurAlpha* et le paramètre de retour vaut toujours 0.
Maximum process Web	Entier long	7	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 -> 32 767 **Description** : Nombre maximum de process Web à maintenir en mode sans contexte avec 4D en mode local et 4D Server. Par défaut, la valeur est 10. Afin que le serveur Web soit réactif, en mode sans contexte, 4D maintient endormis les process Web pendant 5 secondes, et les réutilise pour traiter les éventuelles requêtes HTTP suivantes. En termes de performances, ce principe est en effet plus avantageux que la création d'un nouveau process à chaque requête. Une fois un process Web réutilisé, il est à nouveau endormi pour 5 secondes, sauf si le nombre maximum de process Web est dépassé (auquel cas il est tué). Si aucune requête n'a été attribuée à un process Web durant les 5 secondes, il est tué, sauf si un nombre minimum de process Web a été fixé et est atteint (auquel cas il est à nouveau endormi). Ces paramètres vous permettent d'ajuster le fonctionnement de votre serveur Web en fonction du nombre de

Constante	Type	Valeur	Comment
			tailles de la mémoire disponible, etc.
Minimum process Web	Entier long	6	**Portée** : 4D local, 4D Server **Conservé entre deux sessions :** Oui **Valeurs possibles** : 0 -> 32 767 **Description** : Nombre minimum de process Web à maintenir en mode sans contexte avec 4D en mode local et 4D Server. Par défaut, la valeur est 0 (cf. ci-dessous).
Mode Unicode	Entier long	41	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (mode compatibilité) ou 1 (mode Unicode) **Description** : Mode d'exécution courant de la base, relatif au jeu de caractères. 4D prend en charge le jeu de caractères Unicode mais peut fonctionner en mode "compatibilité" (basé sur le jeu de caractères ASCII Mac). Par défaut, les bases de données converties sont exécutées en mode compatibilité (0) et les bases créées à partir de la version 11 sont exécutées en mode Unicode. Le mode d'exécution est contrôlé via une option des préférences et peut également être lu ou (à des fins de test) modifié via ce sélecteur. La modification de cette option nécessite le redémarrage de la base pour être prise en compte. A noter que, dans le cadre d'un composant, il n'est pas possible de modifier cette valeur, mais uniquement de la lire.
Niveau de compression HTTP	Entier long	50	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Nombre de formules en cache	Entier long	92	**Portée** : Application 4D. **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif **Valeur par défaut** : 0 (pas de cache) **Description** : Fixe ou lit le nombre maximum de formules à conserver dans le cache des formules, qui est utilisé par la commande **EXECUTER FORMULE**. Cette limite est appliquée à tous les process, mais chaque process dispose de son propre cache de formules. Placer des formules dans le cache accélère l'exécution de la commande **EXECUTER FORMULE** en mode compilé puisque chaque formule en cache est tokenisée une seule fois dans ce cas. Lorsque vous modifiez la valeur du cache, son contenu est réinitialisé même si la nouvelle valeur est supérieure à la précédente. Une fois le nombre maximum de formules en cache atteint, toute nouvelle formule exécutée écrase la plus ancienne dans le cache (mode FIFO). Ce paramètre est pris en compte uniquement dans les bases ou les composants compilés.
Numéro automatique table	Entier long	31	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Valeurs possibles** : Toute valeur de type entier long. **Description** : Ce sélecteur permet de modifier ou de lire le numéro unique courant des enregistrements de la table passée en paramètre. "Numéro courant" signifie "dernier numéro utilisé" : si vous modifiez cette valeur à l'aide de **FIXER PARAMETRE BASE**, le prochain enregistrement sera créé avec comme numéro la valeur passée + 1. Ce nouveau numéro est, lui, retourné par la commande **Numerotation automatique** ainsi que dans tout champ de la table auquel la propriété "Incrémentation auto" a été affectée en Structure ou via le SQL. Par défaut, le numéro unique est défini par 4D et correspond à l'ordre de création des enregistrements. Pour des informations supplémentaires, reportez-vous à la documentation de la commande **Numerotation automatique**.
Numéro de port HTTPS	Entier long	39	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Numéro de port serveur SQL	Entier long	88	**Portée** : 4D mode local et 4D Server. **Conservé entre deux sessions** : Oui **Description** : Permet de lire ou de fixer le numéro du port TCP utilisé par le serveur SQL intégré de 4D en mode local ou de 4D Server. Par défaut, la valeur est 19812. Le numéro de port TCP peut également être défini dans la page "SQL" de la boîte de dialogue des Propriétés de la base. Lorsque ce sélecteur est utilisé en écriture, la propriété de la base est mise à jour. **Valeurs possibles** : 0 à 65535. **Valeur par défaut** : 19812
Numéro du port	Entier long	15	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Non **Description** : Numéro du port TCP utilisé par le serveur Web 4D avec 4D en mode local et 4D Server. Par défaut, la valeur est 80. Le numéro de port TCP est défini dans la page "Web/Configuration" de la boîte de dialogue des Propriétés de la base. Vous pouvez utiliser les constantes du thème **Numéros de port TCP** pour le paramètre *valeur*. Le sélecteur Numéro du port est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d'accès au mode Développement). Pour plus d'informations sur le numéro de port TCP, reportez-vous à la section **Paramétrages du serveur Web**.
Numéro du port client serveur	Entier long	35	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 à 65535 **Description** : Numéro de port TCP sur lequel 4D Server publie la base de données (à destination des postes 4D distants). Par défaut, la valeur est 19813. La personnalisation de cette valeur permet d'utiliser plusieurs applications 4D client-serveur sur la même machine avec le protocole TCP ; dans ce cas, vous devez spécifier un numéro de port différent pour chaque application. La valeur est stockée dans le fichier de structure de la base. Elle peut être définie avec 4D en mode local mais n'est prise en compte qu'en configuration client-serveur. Lorsque vous modifiez cette valeur, il est nécessaire de redémarrer le poste serveur afin que la nouvelle valeur soit prise en compte.
Périodicité écriture cache	Entier long	95	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Non **Valeurs possibles** : entier long > 1 (secondes) **Description** : Permet de lire ou de fixer la valeur courante de périodicité de l'écriture du cache de données sur le

Constante	Type	Valeur	Commentaire
			disque, exprimée en secondes. Si elle est modifiée, cette valeur remplace la valeur définie par l'option **Ecriture cache toutes les <n> secondes/minutes** dans la **Page Base de données/Mémoire** des Propriétés de la base durant la session courante (elle n'est pas stockée dans les Propriétés de la base). **Portée** : Application 4D **Conservé entre deux sessions** : Non
PHP adresse IP interpréteur	Entier long	55	**Valeurs** : Chaîne formatée du type "nnn.nnn.nnn.nnn" (par exemple "127.0.0.1"). **Description** : Adresse IP utilisée localement par 4D pour communiquer avec l'interpéteur PHP via fastcgi. Par défaut, la valeur est "127.0.0.1". Cette adresse doit correspondre à la machine sur laquelle se trouve 4D. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations sur l'interpréteur PHP de 4D, reportez-vous au manuel *Mode Développement*.
PHP nombre enfants	Entier long	57	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Valeur de type entier long positif. Par défaut, la valeur est 5. **Description** : Nombre de process enfants à créer et à maintenir localement par l'interpréteur PHP de 4D. Pour des raisons d'optimisation, l'interpréteur PHP crée et utilise un ensemble (pool) de process système appelés "process enfants" pour traiter les demandes d'exécution de scripts. Vous pouvez faire varier le nombre de process enfants en fonction des besoins de votre application. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations sur l'interpréteur PHP de 4D, reportez-vous au manuel *Mode Développement*. **Note :** Sous Mac OS, tous les process enfants partagent le même port. Sous Windows, chaque process enfant utilise un numéro de port spécifique. Le premier numéro est celui défini pour l'interpréteur PHP, les autres process enfants l'incrémentent. Par exemple, si le port par défaut est le 8002 et que vous lancez 5 process enfants, ils utiliseront les ports 8002 à 8006.
PHP nombre requêtes max	Entier long	58	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Valeur de type entier long positif. Par défaut, la valeur est 500. **Description** : Nombre maximum de requêtes acceptées par l'interpréteur PHP. Lorsque ce nombre maximum est atteint, l'interpréteur retourne des erreurs du type "serveur occupé". Pour des raisons de sécurité ou de performance, vous pouvez modifier cette valeur. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations, reportez-vous à la documentation de fastcgi-php. **Note :** Côté 4D, ces paramètres sont appliqués dynamiquement, il n'est pas nécessaire de quitter le programme pour leur prise en compte. En revanche, si l'interpréteur PHP était déjà lancé, il est nécessaire de le quitter et de le relancer pour qu'il prenne en compte ces modifications.
PHP port interpréteur	Entier long	56	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Valeur de type entier long positif. Par défaut, la valeur est 8002. **Description** : Numéro du port TCP utilisé par l'interpréteur PHP de 4D. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations sur l'interpréteur PHP de 4D, reportez-vous au manuel *Mode Développement*.
PHP utiliser interpréteur externe	Entier long	60	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : 0 = utiliser interpréteur interne, 1 = utiliser interpréteur externe **Description** : Valeur indiquant si les requêtes PHP de 4D sont adressées à l'interpréteur interne fourni par 4D ou un interpréteur externe. Par défaut, la valeur est 0 (utilisation de l'interpréteur fourni par 4D). Si vous souhaitez utiliser votre propre interpréteur PHP, par exemple pour bénéficier de modules supplémentaires ou d'une configuration spécifique, passez 1 dans *valeur*. Dans ce cas, 4D ne démarrera pas son interpréteur en cas de requête PHP. L'interpréteur PHP personnalisé doit avoir été compilé en fastcgi et se trouver sur la même machine que le moteur 4D. A noter que dans ce cas, vous devez entièrement gérer l'interpréteur, il n'est ni démarré ni stoppé par 4D. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base.
Prise en charge QuickTime	Entier long	82	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (défaut) = QuickTime désactivé, 1 = QuickTime activé. **Description** : Dans 4D à compter de la v14, par défaut les codecs QuickTime ne sont plus pris en charge. Par compatibilité, vous pouvez les réactiver dans votre base à l'aide de ce sélecteur. La modification de cette option nécessite le redémarrage de la base. A noter toutefois que la prise en charge de QuickTime sera définitivement supprimée dans les prochaines versions de 4D.
Process Web simultanés maxi	Entier long	18	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Seuil de compression HTTP	Entier long	51	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
SQL autocommit	Entier long	43	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (désactivation) ou 1 (activation) **Description** : Activation ou désactivation du mode SQL auto-commit. Par défaut, la valeur est 0 (mode désactivé) Le mode auto-commit permet de renforcer l'intégrité référentielle de la base. Lorsque ce mode est actif, les requêtes **SELECT**, **INSERT**, **UPDATE**, **DELETE** (SIUD) sont automatiquement incluses dans des transactions lorsqu'elles sont exécutées en-dehors de toute transaction. Ce mode peut également être défini dans les préférences de la base. **Portée** : Poste 4D distant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (pas de synchronisation), 1 (synchronisation auto) ou 2 (demander). **Description** : Mode de synchronisation dynamique du dossier *Resources* du poste client 4D ayant exécuté la commande avec celui du serveur. Lorsque le contenu du dossier *Resources* sur le serveur a été modifié ou qu'une demande de synchronisation a été émise (via l'explorateur de ressources ou suite à l'exécution de la commande **NOTIFIER MODIFICATION**

Constante	Type	Valeur	Comment
Synchro auto dossier Resources	Entier long	48	**DOSSIER RESOURCES**), le serveur notifie les clients connectés. Trois modes de synchronisation sont alors possibles côté client. Le sélecteur Synchro auto dossier Resources vous permet de définir le mode à utiliser pour le poste client et la session courante : • 0 (valeur par défaut) : pas de synchronisation dynamique (la demande de synchronisation est ignorée) • 1 : synchronisation dynamique automatique • 2 : affichage d'une boîte de dialogue sur les postes clients, avec possibilité d'effectuer ou de refuser la synchronisation. Le mode de synchronisation peut également être défini globalement dans les Propriétés de la base.
Taille maxi mémoire temporaire	Entier long	61	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif. **Description** : Taille maximum de mémoire temporaire que 4D pourra allouer à chaque process, exprimée en Mo. Par défaut, la valeur est 0 (pas de taille maximum). 4D utilise une mémoire temporaire spéciale dédiée aux opérations d'indexation et de tri. Cette mémoire a pour but de préserver la mémoire cache "classique" lors d'opérations massives. Elle n'est activée qu'en cas de besoin. Par défaut, la taille de la mémoire temporaire n'est limitée que par les ressources disponibles (en fonction de la configuration mémoire du système). Ce mécanisme convient à la plupart des applications. Toutefois, dans certains contextes spécifiques, notamment lorsqu'une application client-serveur effectue simultanément un grand nombre de tris séquentiels, la taille de la mémoire temporaire peut augmenter de façon critique, jusqu'à rendre le système instable. Dans ce contexte, fixer une taille maximum à la mémoire temporaire permet de préserver le fonctionnement de l'application. En contrepartie, la vitesse d'exécution pourra être affectée : lorsque la taille maximum est atteinte pour un process, 4D utilise des fichiers disque, ce qui peut ralentir les traitements. Pour des besoins tels que ceux décrits ci-dessus, une taille maximum d'environ 50 Mo est généralement un bon compromis. La valeur idéale sera cependant à déterminer en fonction des spécificités de l'application et résultera généralement de tests en volumétrie réelle.
Taille maximum requêtes Web	Entier long	27	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Taille minimum libération cache	Entier long	66	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif > 1. **Description** : Taille minimum de mémoire à libérer du cache de la base de données lorsque le moteur a besoin d'y faire de la place pour y allouer un objet (valeur en octets). Ce sélecteur a pour but de permettre de réduire le nombre de libérations de données du cache afin d'obtenir des gains de performances. Vous pouvez faire varier ce paramétrage en fonction de la taille du cache et de celle des blocs de données manipulées dans votre base. Par défaut, si ce sélecteur n'est pas utilisé, 4D décharge au minimum 10 % du cache en cas de besoin de place.
Taille pile process base server	Entier long	53	**Portée** : 4D Server **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif. **Description** : Taille de la pile allouée à chaque process système préemptif sur le serveur, exprimée en octets. La taille par défaut est déterminée par le système. Les process système préemptifs (process de type Process base 4D client) sont chargés de contrôler les process clients 4D principaux. La taille allouée par défaut à la pile de chaque process préemptif permet un bon confort d'exécution mais peut s'avérer conséquente lorsque de très nombreux process (plusieurs centaines) sont créés. A des fins d'optimisation, cette taille peut être diminuée sensiblement si les opérations effectuées par la base s'y prêtent (par exemple si la base n'effectue pas de tris sur de grosses quantités d'enregistrements). Des valeurs de 512 voire de 256 Ko sont possibles. Attention, le sous-dimensionnement de la pile est critique et peut nuire au fonctionnement de 4D Server. Le réglage de ce paramètre est à effectuer avec précaution et doit tenir compte des conditions d'utilisation de la base (nombre d'enregistrements, types d'opérations, etc.). Pour être pris en compte, ce paramétrage doit être exécuté sur le poste serveur (par exemple dans la méthode base Sur démarrage serveur).
Timeout 4D mode distant	Entier long	14	**Portée** (ancienne couche réseau uniquement) : Application 4D si *valeur* positive **Conservé entre deux sessions** : Oui si *valeur* positive **Description** : A utiliser dans des cas très spécifiques. Valeur du délai avant déconnexion (timeout) accordé par le poste 4D distant au poste 4D Server. Par défaut, cette valeur est définie dans la page "Client-Serveur/Options réseau" des Propriétés de la base, sur le poste distant. Le sélecteur Timeout 4D mode distant n'est pris en compte que si vous utilisez l'ancienne couche réseau. Avec la couche *ServerNet* activée, il est ignoré : ce paramétrage est entièrement géré par le sélecteur Timeout 4D Server (13).
Timeout 4D Server	Entier long	13	**Portée** : Application 4D si *valeur* positive **Conservé entre deux sessions** : Oui si *valeur* positive **Valeurs possibles** : 0 -> 32 767 **Description** : Valeur du délai avant déconnexion (timeout) accordé par 4D Server aux postes clients. Par défaut, cette valeur est définie dans la page "Client-Serveur/Options réseau" des Propriétés de la base, sur le poste serveur. Le timeout serveur définit la période maximale de non-réponse du client "autorisée", par exemple s'il effectue une opération bloquante. A l'issue de cette période, 4D Server déconnecte le client. Le sélecteur Timeout 4D Server vous permet de fixer un nouveau timeout, exprimé en minutes. Cette possibilité permet en particulier d'augmenter la valeur du timeout avant l'exécution sur le poste client d'une opération bloquante de longue durée, risquant d'entraîner une déconnexion ; par exemple, l'impression d'un grand nombre de pages. Vous disposez en outre de deux possibilités : • effectuer une modification globale et permanente : la nouvelle valeur s'applique à tous les process et est stockée dans les préférences de l'application (équivaut à une modification de la valeur dans la boîte de dialogue des Préférences). Pour cela, passez une valeur **positive** dans le paramètre *valeur*. • effectuer une modification restreinte et temporaire : la nouvelle valeur ne s'applique qu'au process appelant (les autres process conservant la valeur d'origine), et est abandonnée dès que le serveur reçoit un signe d'activité du poste client — par exemple, dès que l'opération est terminée. Cette possibilité est utile pour gérer les opérations longues initiées par des plug-ins. Pour cela, passez une valeur **négative** dans le

Constante	Type	Valeur	Comment
Timeout connexions inactives	Entier long	54	paramètre *valeur*. Pour définir une connexion “Ouverte en permanence”, passez 0 dans *valeur*. Reportez-vous à l’exemple 1. **Portée** : Application 4D sauf si valeur négative **Conservé entre deux sessions** : Non **Valeurs possibles** : Valeur entière exprimant une durée en secondes. La valeur peut être positive (nouvelles connexions) ou négative (connexions existantes). Par défaut, la valeur est 20. **Description** : Délai maximum d’inactivité (timeout) des connexions au moteur de base de données et au serveur SQL de 4D ainsi que, en mode *ServerNet* (nouvelle couche réseau), au serveur d'applications 4D. Lorsqu’une connexion inactive atteint ce délai, elle est automatiquement mise en veille, ce qui se traduit par le gel de la session client/serveur et la fermeture du socket réseau. Dans la fenêtre d'administration du serveur, le process utilisateur prend l'état "Postponed". Ce fonctionnement est entièrement transparent pour l’utilisateur : dès qu’il y a reprise d’activité sur la connexion mise en veille, le socket est automatiquement rouvert et la session client/serveur restaurée. Ce paramétrage permet, d’une part, d’économiser des ressources sur le serveur : les connexions mises en veille referment le socket et libèrent un processus sur le serveur. D’autre part, il permet d’éviter les pertes de connexions dues aux fermetures par les pare-feux des sockets inactifs. La valeur de timeout des connexions inactives doit pour cela être inférieure à celle du pare-feu. Si vous passez une valeur positive dans *valeur*, elle s'applique à toutes les nouvelles connexions dans tous les process. Si vous passez une valeur négative, elle s’applique aux connexions ouvertes dans le process courant. Si vous passez 0, les connexions inactives ne sont pas soumises à un timeout. Ce paramètre peut être défini côté serveur et côté client. Si vous passez deux durées différentes, la plus courte sera prise en compte. Généralement, vous n'aurez pas besoin de modifier cette valeur.
Trier par formule serveur	Entier long	47	**Portée** : Table et process courants **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (utiliser le paramétrage de la base), 1 (exécuter sur le client) ou 2 (exécuter sur le serveur) **Description** : Emplacement de l’exécution de la commande **TRIER PAR FORMULE** pour la table passée en paramètre. Dans le cadre de l’exploitation d’une base en client-serveur, la commande **TRIER PAR FORMULE** peut être exécutée soit sur le serveur soit sur le client. Ce sélecteur permet de définir l’emplacement de l’exécution de cette commande (serveur ou client). Ce mode peut également être défini dans les préférences de la base. Pour plus d’informations, reportez-vous à la description du sélecteur 46, Chercher par formule serveur. **Note :** Si vous souhaitez pouvoir activer les jointures "type SQL" (cf. sélecteur Jointures chercher par formule), vous devez toujours exécuter les formules sur le serveur afin qu'elle ait accès aux enregistrements. Attention, dans ce contexte, la formule ne doit pas contenir d'appel à une méthode, sinon elle est automatiquement basculée sur le poste distant.
Utiliser ancienne couche réseau	Entier long	87	**Portée :** 4D local, 4D Server. **Conservé entre deux sessions :** Oui **Description :** Fixe ou lit le statut courant de l'ancienne couche réseau pour les connexions client/serveur. L'ancienne couche réseau est obsolète à compter de 4D v14 R5 et doit être progressivement remplacée dans vos applications par la couche réseau *ServerNet*. *ServerNet* sera nécessaire dans les prochaines versions de 4D afin de permettre aux applications 4D de tirer parti des futures évolutions réseau. Pour des raisons de compatibilité, l'ancienne couche réseau est toujours prise en charge afin de faciliter la transition des applications existantes (elle reste utilisée par défaut dans les applications converties depuis des versions antérieures à la v14 R5). Passez 1 dans ce paramètre pour utiliser l'ancienne couche réseau (et désactiver *ServerNet*), et passez 0 pour désactiver l'ancienne couche réseau (et utiliser *ServerNet*). Cette propriété peut également être définie à l'aide de l'option "Utiliser l'ancienne couche réseau" présente dans la **Page Compatibilité** des Propriétés de la base (voir section **Options réseau et Client-serveur** ; dans cette section, vous trouverez aussi un paragraphe décrivant la stratégie de migration. Nous vous recommandons d'activer *ServerNet* dès que possible). Il est nécessaire de redémarrer l'application pour que ce paramètre soit pris en compte. Cette option n'est pas disponible dans 4D Server v14 R5 version 64 bits pour OS X, qui prend en charge uniquement *ServetNet* (elle vaut toujours 0). **Valeurs possibles :** 0 ou 1 (0 = ne pas utiliser l'ancienne couche, 1 = utiliser l'ancienne couche) **Valeur par défaut :** 0 dans les applications créées avec 4D v14 R5 ou suivantes, 1 dans les applications converties depuis 4D v14 R4 ou précédentes.

Note : Le paramètre *laTable* est utilisé par les sélecteurs 31, 46 et 47 uniquement. Dans les autres cas, il est ignoré s'il est passé.

Exemple 1

L’instruction suivante permet d'anticiper un éventuel problème de **timeout**

```
 `Augmentation du timeout à 3 heures pour le process courant
FIXER PARAMETRE BASE(Timeout 4D Server;-60*3)
 `Exécution d’une opération longue hors du contrôle de 4D
...
WR MAILING(LaZone;3;0)
...
```

Exemple 2

Cet exemple force temporairement l’exécution sur le client d’une commande de recherche par formule :

```
valCourante:=Lire parametre base([table1];Chercher par formule serveur) `Stocker le paramétrage courant
FIXER PARAMETRE BASE([table1];Chercher par formule serveur;1) `Forcer l’exécution sur le client
CHERCHER PAR FORMULE([table1];maformule)
FIXER PARAMETRE BASE([table1];Chercher par formule serveur;valCourante) `Rétablir le paramétrage courant
```

Exemple 3

Vous souhaitez exporter des données en JSON contenant une date 4D convertie. A noter que la conversion a lieu au moment du stockage de la date dans l'objet, il faut donc appeler la commande **FIXER PARAMETRE BASE** avant **OB FIXER** :

```
C_OBJET($o)
FIXER PARAMETRE BASE(JSON fuseau horaire local;0)
OB FIXER($o ;"maDate";Date du jour) // conversion JSON
$json:=JSON Stringify($o)
FIXER PARAMETRE BASE(JSON fuseau horaire local;1)
```

GENERER APPLICATION

GENERER APPLICATION {(nomProjet)}

Paramètre	Type		Description
nomProjet	Chaîne	⇨	Chemin d'accès complet du projet à utiliser

Description

La commande **GENERER APPLICATION** lance le processus de génération d'application en prenant en compte les paramètres définis dans le projet d'application courant ou le projet d'application désigné par le paramètre *nomProjet*.

Un projet d'application est un fichier XML contenant tous les paramétrages utilisés pour générer une application. La plupart de ces paramétrages sont visibles dans la boîte de dialogue du Générateur d'application (pour plus d'informations sur le Générateur d'application, reportez-vous à la section **Générateur d'applications** dans le manuel Mode Développement de 4D).
Par défaut, 4D crée pour chaque base de données un projet d'application par défaut nommé "buildapp.xml" et placé dans le sous-dossier BuildApp du dossier Preferences de la base.

Si la base n'a pas été compilée ou si le code compilé n'est pas à jour, la commande lance au préalable le processus de compilation. Dans ce cas, la fenêtre du compilateur n'apparaît pas (sauf en cas d'erreur), seule une barre de progression est affichée.
Vous pouvez éviter l'affichage de cette barre de progression à l'aide de la commande **SUPPRIMER MESSAGES**.

Si vous ne passez pas le paramètre facultatif *nomProjet*, la commande affiche une boîte de dialogue standard d'ouverture de document, vous permettant de désigner un fichier de projet. La variable système Document contiendra le chemin d'accès complet du fichier sélectionné. Si vous passez le chemin d'accès et le nom d'un fichier XML de projet d'application valide (encodage UTF-8 et extension ".xml"), la commande utilisera les paramètres définis dans le fichier. Pour plus d'informations sur la structure et les clés utilisables dans un fichier XML de projet d'application, reportez-vous au manuel **4D Clés XML BuildApplication**.

Exemple

Génération de deux applications dans une seule méthode :

```
GENERER APPLICATION("c:\\dossier\\projets\\monprojet1.xml")
Si(OK=1)
   GENERER APPLICATION("c:\\dossier\\projets\\monprojet2.xml")
Fin de si
```

Variables et ensembles système

La variable système OK prend la valeur 1 si la commande a été correctement exécutée, sinon elle prend la valeur 0. La variable système Document prend le chemin d'accès complet du fichier de projet ouvert.

Gestion des erreurs

Si la commande échoue, une erreur est générée, que vous pouvez intercepter à l'aide de la commande **APPELER SUR ERREUR**.

Lire chemin journal derniere mise a jour

Lire chemin journal derniere mise a jour -> Résultat

Paramètre	Type		Description
Résultat	Texte		Chemin d'accès du journal de mise à jour le plus récent

Description

La commande **Lire chemin journal derniere mise a jour** retourne le chemin d'accès complet du fichier journal de mise à jour le plus récent sur le poste où elle est appelée.

Le journal de mise à jour est généré par 4D lors des processus de mise à jour automatiques. Il contient les informations relatives aux mises à jour effectuées ainsi que les éventuelles erreurs rencontrées.

Cette commande est destinée à être utilisée dans un processus de mise à jour automatique d'une application fusionnée (serveur ou monoposte). Pour plus d'informations, reportez-vous à la section **Finaliser et déployer les applications finales** dans le manuel *Mode Développement*.

Lire fragmentation table

Lire fragmentation table (laTable) -> Résultat

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle connaître le taux de fragmentation
Résultat	Réel	⇦	Pourcentage de fragmentation

Description

La commande **Lire fragmentation table** retourne le pourcentage de fragmentation logique des enregistrements de la table désignée par le paramètre *laTable*.

Le taux de fragmentation logique des enregistrements indique si les enregistrements sont stockés de manière ordonnée dans le fichier de données. Une fragmentation trop élevée peut ralentir sensiblement les tris et les recherches séquentiels sur la table. Un pourcentage de fragmentation de 0 correspond à une fragmentation nulle. Au-delà de 20 %, il peut être intéressant de procéder au compactage du fichier de données.

Exemple

Cette méthode de maintenance permet de demander le compactage du fichier de données en cas de fragmentation importante d'au moins une table de la base :

```
ACompacter:=Faux
Boucle($i;1;Lire numero derniere table)
      Si(Est un numero de table valide($i))
         Si(Lire fragmentation table(Table($i)->)>20)
            ACompacter:=Vrai
         Fin de si
      Fin de si
Fin de boucle
Si(ACompacter)
       // Poser un marqueur de demande de compactage
Fin de si
```

LIRE INFORMATIONS SERIALISATION

LIRE INFORMATIONS SERIALISATION (clé ; nomUtilisateur ; société ; connectés ; maxUtilisateurs)

Paramètre	Type		Description
clé	Entier long	⇦	Clé unique du produit (crypté)
nomUtilisateur	Chaîne	⇦	Nom enregistré
société	Chaîne	⇦	Organisation enregistrée
connectés	Entier long	⇦	Nombre d'utilisateurs connectés
maxUtilisateurs	Entier long	⇦	Nombre maximum d'utilisateurs

Description

La commande **LIRE INFORMATIONS SERIALISATION** retourne diverses informations relatives à la sérialisation de l'application 4D courante.

- *clé* : identifiant unique du produit installé. Ce numéro unique correspond à une seule application 4D (4D Server, 4D en mode local, 4D Desktop, etc.) installée sur un seul poste. Bien entendu, ce numéro est crypté.
- *utilisateur* : Nom de l'utilisateur de l'application, tel qu'il a été saisi au moment de l'installation.
- *société* : Nom de la société ou de l'organisation à laquelle appartient l'utilisateur, tel qu'il a été saisi au moment de l'installation.
- *connectés* : Nombre d'utilisateurs connectés au moment de l'exécution de la commande.
- *maxUtilisateurs* : Nombre maximal d'utilisateurs pouvant se connecter simultanément.

Note : Les deux derniers paramètres retournent toujours 1 pour les versions monopostes de 4D, sauf lorsqu'il s'agit de versions de démonstration, auquel cas ils retournent 0.

La commande **LIRE INFORMATIONS SERIALISATION** s'inscrit dans le schéma général de protection des composants proposé par 4D.

Les développeurs de composants peuvent, s'ils le souhaitent, lier chaque copie de leur produit à une seule application 4D installée, afin d'empêcher toute copie illicite.

Le principe de fonctionnement du système est le suivant : un utilisateur souhaitant acquérir un composant fournit au développeur sa clé unique — générée à l'aide de la commande **LIRE INFORMATIONS SERIALISATION**. Cette opération peut, par exemple, être effectuée par l'intermédiaire d'un formulaire "Bon de commande" intégré à la version de démonstration du composant. La clé générée est unique : il n'existe qu'une clé par application 4D installée.
Le développeur du composant peut alors générer son propre numéro de série, en combinant la clé et l'algorithme de cryptage de son choix. Le composant livré comportera une fonction permettant de tester si les informations retournées par **LIRE INFORMATIONS SERIALISATION** correspondent bien à ce numéro de série. Dans le cas contraire, le composant sera rendu inutilisable.

Note : Les développeurs de plug-ins peuvent également bénéficier de ce système de protection. Pour plus d'informations, reportez-vous à la documentation de 4D Plugin Kit.

Lire langue base

Lire langue base {(typeLangue)} -> Résultat

Paramètre	Type		Description
typeLangue	Entier long	⇨	Type de langue
Résultat	Chaîne	⇦	Code de la langue utilisée

Description

La commande **Lire langue base** retourne la langue par défaut ou la langue désignée par *typeLangue* de la base, exprimée dans la norme définie par la RFC 3066. Typiquement, la commande retourne “fr” pour le français “es” pour l’espagnol, etc. Pour plus d’informations sur cette norme et sur les valeurs retournées par cette commande, reportez-vous à l'**MissingRef** dans le manuel *Mode Développement*.

Plusieurs paramétrages de langues différents peuvent être utilisés simultanément dans l’application. Pour désigner le paramétrage à obtenir, passez dans *typeLangue* une des constantes suivantes, placées dans le thème **Environnement 4D** :

Constante	Type	Valeur	Comment
Langue 4D interne	Entier long	3	Langue utilisée par 4D pour les tris et les comparaisons de textes (définie dans les Préférences de l’application).
Langue courante	Entier long	1	Langue courante de l’application : langue par défaut ou langue définie via la commande **FIXER LANGUE BASE**.
Langue par défaut	Entier long	0	Langue définie automatiquement par 4D au démarrage en fonction du dossier Resources et de l’environnement système (non modifiable).
Langue système	Entier long	2	Langue définie par l’utilisateur courant du système.

Par défaut, si vous omettez le paramètre *typeLangue*, la commande retourne la langue par défaut (0).

La langue courante de la base permet de définir le dossier .lproj dans lequel le programme va chercher les éléments localisés de la base de données. 4D détermine automatiquement la langue courante au démarrage de la base en fonction du contenu du dossier **Resources** et de l’environnement système. Le principe est que 4D charge le premier dossier .lproj de la base correspondant à la langue de référence, dans l’ordre de priorité suivant :

1. Langue du système (sous Mac OS, plusieurs langues peuvent être définies avec un ordre de préférence, 4D utilise ce paramétrage).
2. Langue de l’application 4D.
3. Anglais
4. Première langue trouvée dans le dossier **Resources**.

Note : Si la base ne contient aucun dossier .lproj, 4D applique l'ordre de priorité suivant : 1. Langue du système, 2. Anglais (si la langue du système n'a pas pu être identifiée).

Lire mesures base

Lire mesures base {{ options }} -> Résultat

Paramètre	Type		Description
options	Objet	⇒	Options de retour
Résultat	Objet	⇐	Objet contenant des mesures sur la base

Description

La commande **Lire mesures base** vous permet d'obtenir un ensemble d'informations détaillées sur les événements du moteur de base de données de 4D. Les informations renvoyées concernent les accès en lecture/écriture aux données depuis ou vers le disque ou le cache ainsi que l'utilisation des index de la base, les recherches et les tris.

Lire mesures base retourne un seul objet contenant toutes les mesures utiles. Le paramètre *options* vous permet de paramétrer les informations retournées.

Présentation de l'objet retourné

L'objet retourné par la commande contient une seule propriété, nommée "DB", dont la structure est la suivante :

```
{
    "DB": {
        "diskReadBytes": {…},
        "cacheReadBytes": {…},
        "cacheMissBytes": {…},
        "diskWriteBytes": {…},

        "diskReadCount": {…},
        "cacheReadCount": {…},
        "cacheMissCount": {…},
        "diskWriteCount": {…},

        "dataSegment1": {…},
        "indexSegment": {…},

        "tables": {…},
        "indexes": {…}

    }
}
```

Cet objet est composé de huit propriétés élémentaires qui contiennent les mesures de base ("diskReadBytes", "cacheReadBytes", "cacheMissBytes", "diskWriteBytes", "diskReadCount", "cacheReadCount", "cacheMissCount", "diskWriteCount") ainsi que des propriétés additionnelles ("dataSegment1", "indexSegment", "tables", "index") qui peuvent elles-mêmes contenir les propriétés élémentaires mais appliquées à des niveaux différents (voir ci-dessous).

Note : Une propriété est présente dans l'objet uniquement si elle contient des valeurs. Lorsqu'une propriété est vide, elle n'est pas incluse dans l'objet. Par exemple, si la base a été ouverte en mode lecture seulement et que les index n'ont pas été sollicités, l'objet retourné ne contiendra pas les propriétés "diskWriteBytes", "diskWriteCount", "indexSegment" et "indexes".

Propriétés élémentaires

Les propriétés élémentaires peuvent être présentes à différents niveaux de l'objet DB. Elles retournent les mêmes informations mais sur des périmètres spécifiques. Voici la description de ces propriétés :

Nom	Information retournée
diskReadBytes	Octets lus depuis le disque
cacheReadBytes	Octets lus depuis le cache
cacheMissBytes	Octets manqués depuis le cache
diskWriteBytes	Octets écrits sur le disque
diskReadCount	Nombre d'accès en lecture depuis le disque
cacheReadCount	Nombre d'accès en lecture depuis le cache
cacheMissCount	Nombre d'accès manqués dans le cache
diskWriteCount	Nombre d'accès en écriture sur le disque

Ces huit propriétés élémentaires ont toutes la même structure d'objet, par exemple :

```
"diskReadBytes": {     "value": 33486473620,     "history": [        // optionnel         {"value": 52564,"time": -1665},
{"value": 54202,"time": -1649},             …     ] }
```

- "**value**" (numérique): La propriété "value" contient un nombre représentant soit une quantité d'octets, soit un nombre d'accès. Cette valeur représente la somme théorique des valeurs de l'objet "history" (même si l'objet "history" n'est pas présent).
- "**history**" (tableau d'objets) : Le tableau d'objets "history" est une compilation de valeurs d'événements groupés par seconde. La propriété "history" est présente uniquement si elle a été demandée via le paramètre *options* (cf. ci-dessous). Le tableau "history" contient un maximum de 200 éléments. Chaque élément du tableau est lui-même un objet contenant deux propriétés : "value" et "time".
  - "value" (numérique) : nombre d'octets ou d'accès décomptés durant la période de temps indiquée par la propriété "time" associée.
  - "time" (numérique) : nombre de secondes écoulées depuis l'appel de la fonction. Dans l'exemple ci-dessus, ("time": -1649) signifie "il y a 1649 secondes" (ou plus précisément, entre 1649 et 1650 secondes). Pendant cette unique seconde, 54,202 octets ont été lus sur le disque.
    Le tableau "history" ne contient pas séquentiellement toutes les secondes (-1650,-1651,-1652, etc.). La valeur précédente est -1665, ce qui signifie que rien n'a été lu sur le disque durant la période de 15 secondes entre 1650 et 1665.
    Puisque la taille maximum du tableau est 200, si la base de données est sollicitée de manière intensive (quelque chose est lu chaque seconde

sur le disque), la durée maximale de l'historique sera de 200 secondes. D'un autre côté, s'il ne se passe presque rien, par exemple uniquement toutes les 3 minutes, la durée de l'historique pourra atteindre 600 minutes (3*200).
Cet exemple peut être représenté dans le schéma suivant :

4D internal history		Requested history: 30	
time	value	time	value
-2	4629	0	0
-4	7788	-1	0
-6	3718	-2	4629
-8	8814	-3	0
-10	3925	-4	7788
-12	775	-5	0
-14	6807	-6	3718
-16	3265	-7	0
-18	8086	-8	8814
-20	2539	-9	0
		-10	3925
		-11	0
		-12	775
		-13	0
		-14	6807
		-15	0
		-16	3265
		-17	0
		-18	8086
		-19	0
		-20	2539
		-21	-1
		-22	-1
		-23	-1
		-24	-1
		-25	-1
		-26	-1
		-27	-1
		-28	-1
		-29	-1
		-30	-1

dataSegment1 et indexSegment

Les propriétés "dataSegment1" et "indexSegment" peuvent contenir jusqu'à quatre propriétés élémentaires (le cas échéant) :

```
"dataSegment1": {
    "diskReadBytes": {…},
    "diskWriteBytes": {…},
    "diskReadCount": {…},
    "diskWriteCount": {…}
    },
"indexSegment": {
    "diskReadBytes": {…},
    "diskWriteBytes": {…},
    "diskReadCount": {…},
    "diskWriteCount": {…}
    }
```

Ces propriétés retournent les mêmes informations que les propriétés élémentaires précédemment décrites, mais limitées à chaque fichier de la base :

- "dataSegment1" représente le fichier de données .4dd sur disque
- "indexSegment" représente le fichier d'index .4dx sur disque

Par exemple, vous pouvez obtenir l'objet suivant :

```
{ "DB": { "diskReadBytes": {     "value": 718260     }, "diskReadCount": {     "value": 229     }, "dataSegment1": {
"diskReadBytes": {     "value": 679092     },     "diskReadCount": {     "value": 212     }     }, "indexSegment": {
"diskReadBytes": {     "value": 39168     },     "diskReadCount": {     "value": 17     } }
```

Les valeurs retournées correspondent aux formules suivantes :

diskReadBytes.value = dataSegment1.diskReadBytes.value + indexSegment.diskReadBytes.value
diskWriteBytes.value = dataSegment1.diskWriteBytes.value + indexSegment.diskWriteBytes.value
diskReadCount.value = dataSegment1.diskReadCount.value + indexSegment.diskReadCount.value
diskWriteCount.value = dataSegment1.diskWriteCount.value + indexSegment.diskWriteCount.value

tables

La propriété "tables" contient autant de propriétés qu'il y a de tables ayant été utilisées en lecture ou en écriture depuis l'ouverture de la base. Le nom de chaque propriété est le nom de la table concernée. Par exemple :

```
"tables": {     "Employees": {…}     "Companies": {…}     }
```

Chaque objet "table" contient jusqu' à 12 propriétés :

- Les huit premières propriétés sont les *propriétés élémentaires* (voir ci-dessus) limitées à la table concernée.
- Deux autres propriétés, "records" et "blobs", contiennent également le même ensemble des huit propriétés élémentaires, mais limitées à certains types de champs :
  - La propriété "records" concerne tous les champs de la table (chaînes, dates, numériques, etc.) à l'exception des champs de type texte, image et BLOB.
  - La propriété "blobs" concerne les champs de type texte, image et BLOB de la table.
- Une ou deux propriétés supplémentaires, "fields" et "queries", peuvent également être présentes en fonction des recherches et tris effectués sur la table concernée :
  - La propriété "fields" contient autant de propriétés "nom de champ" (chacune étant également un sous-objet) qu'il y a de champs ayant été utilisés pour des recherches ou des tris.
    Chaque objet nom de champ contient :
    - un objet "queryCount" (avec ou sans history, en fonction du paramètre *options*) si une recherche a été effectuée en utilisant ce champ
    - et/ou un objet "sortCount" (avec ou sans history, en fonction du paramètre *options*) si un tri a été effectué en utilisant ce champ.
    Cet attribut n'est pas basé sur l'utilisation des index ; tous les types de recherches et de tris sont pris en compte.
    Exemple : Depuis le lancement de la base, plusieurs recherches et tris ont été effectués en utilisant les champs *CompID*, *Name* et *FirstName*.
    L'objet retourné contient le sous-objet "fields" suivant (*options* sans historique) :

```
{     "DB": {         "tables": {             "Employees": {                 "fields": {                     "CompID": {
                       "queryCount": {                                 "value": 3                     }
},                     "Name": {                         "queryCount": {                             "value": 1
                       },                          "sortCount": {                                 "value": 3
                       }                         },                      "FirstName": {                          "sortCount": {
                           "value": 2                          }                   } (...)
```

Note : L'attribut "fields" est créé uniquement si une recherche ou un tri a été effectué(e) sur la table ; sinon, l'attribut n'est pas présent.

- "queries" est un tableau d'objets fournissant une description de chaque recherche effectuée sur la table concernée. Chaque élément du tableau contient trois attributs :
  - "queryStatement" (chaîne) : chaîne de recherche (contenant les noms des champs mais pas les valeurs recherchées). Par exemple : "(Companies.PK_ID != ?)"
  - "queryCount" (objet) :
    - "value" (numérique) : nombre d'exécutions de la chaîne de recherche, quelles que soient les valeurs recherchées.
    - "history" (tableau d'objets) (si requis via le paramètre *options*) : propriétés d'historique standard "value" et "time"
  - "duration" (objet) (si la "value" est >0)
    - "value" (numérique) : nombre de millisecondes
    - "history" (tableau d'objets) (si requis via le paramètre *options*) : propriétés d'historique standard "value" et "time".

  Exemple : Depuis le lancement de la base, une seule recherche a été effectuée sur la table Employees (*options* avec historique) :

```
{     "DB": {         "tables": {             "Employees": {                 "queries": [                     {
                       "queryStatement": "(Employees.Name == ?)",                         "queryCount": {
                           "value": 1,                                 "history": [                             {
                                   "value": 1,                                     "time": -2022
                               }                               ]                         },
"duration": {                           "value": 2,                             "history": [
                               {                                   "value": 2,
"time": -2022                                   }                           ]                       }
}, (...)
```

Note : L'attribut "queries" est créé si au moins une recherche a été effectuée sur la table.

indexes

Il s'agit de l'objet ayant la structure la plus complexe. Toutes les tables auxquelles on a accédé via au moins l'un de leurs index sont stockées en tant que propriétés et, à l'intérieur de chaque propriété, les noms des index utilisés sont également stockés sous forme de propriétés. Les index de mots-clés apparaissent séparément, leur nom est suivi de "*(Keyword)*". Enfin, chaque objet nom d'index contient les huit propriétés élémentaires relatives à cet index ainsi que jusqu'à quatre sous-objets en fonction de l'utilisation des index de la base depuis son lancement (chaque sous-objet n'existe que si au moins une opération correspondante a été effectuée depuis le lancement de la base).

Exemple : Depuis le lancement de la base, divers index du champ [Employees]EmpLastName ont été sollicités. En outre, 2 enregistrements ont été créés et 16 enregistrements ont été supprimés dans la table [Companies]. Cette table comporte un champ "name" qui est indexé. Des recherches et des tris ont été effectués dans la table via ce champ. L'objet résultant contient :

```
"indexes": {     "Employees": {         "EmpLastName": {                     "diskReadBytes": {…},
"cacheReadBytes": {…},                    "cacheMissBytes": {…},                     "diskWriteBytes": {…},
"diskReadCount": {…},                   "cacheReadCount": {…},                    "cacheMissCount": {…},
"diskWriteCount": {…}             }         "EmpLastName (Keyword)": {...},         "index3Name": {…},         "index4Name": {…},
    …         }         "Companies": {             "Name":             (...)                 "queryCount": {
"value": 41                 },                "sortCount": {                     "value": 3                 },
"insertKeyCount": {                     "value": 2                },                 "deleteKeyCount": {                     "value":
16                }     table3Name: {…} }
```

Paramètre options

Le paramètre *options* vous permet de personnaliser les informations retournées par la commande. Dans *options*, vous devez passer un objet pouvant contenir jusqu'à trois propriétés : "withHistory", "historyLength" et "path".

Propriété	Type	Description
"withHistory"	Booléen	"true" signifie que l'objet "history" devra être retourné par la commande; "false" signifie que l'objet retourné ne devra pas contenir d'objet "history"
"historyLength"	numérique	Définit la taille en secondes du tableau "history" retourné(*).
"path"	chaîne | tableau de chaînes	Chemin complet de la propriété spécifique ou tableau de chemins complets des propriétés spécifiques que vous voulez obtenir. Si vous passez une chaîne, seule la valeur correspondante est retournée dans l'objet "DB" (si le chemin est valide). Exemple : "DB.tables.Employees.records.diskWriteBytes". Si vous passez un tableau de chaînes, seules les valeurs correspondantes sont retournées dans l'objet "DB" (si les chemins sont valides). Exemple : ["DB.tables.Employee.records.diskWriteBytes", "DB.tables.Employee.records.diskReadCount","DB.dataSegment1.diskReadBytes"]

(*) Comme décrit précédemment, l'historique n'est pas stocké sous forme d'une séquence de secondes mais uniquement sous forme de valeurs remarquables. Si rien ne se produit durant deux secondes ou plus, rien n'est stocké et une rupture apparaît dans le tableau "history". Par exemple, "time" peut contenir -2, -4, -5, -10, -15, -30 avec des valeurs "value" 200, 300, 250, 400, 500,150. Si la propriété "historyLength" est fixée à 600 (10 minutes), le tableau retourné contiendra 0, -1, -2, -3 … -599 pour "time", et seules les propriétés "value" des secondes -2, -4, -5, -10, -15, -30 seront remplies. Toutes les autres propriétés "value" auront pour valeur 0 (zéro). De plus, comme décrit également, la seule limite du tableau d'historique interne est sa taille (200 éléments), et non le temps. Cela signifie que s'il y a une activité réduite pour une propriété spécifique, le moment le plus ancien peut être très éloigné (p.e. -3600 pour il y a une heure). Il peut également contenir moins de 200 valeurs si la base vient juste d'être lancée. Dans ces cas, si l'heure interne de l'historique est plus récent que celui demandé OU si toutes les valeurs remarquables ont déjà ajoutées au tableau retourné, la valeur retournée sera -1. Exemple : La base a été démarrée il y a 20 secondes et la taille demandée du tableau "history" est de 60 secondes. Les données retournées entre 0 et -20 seront bien constituées de valeurs et de 0, et les autres valeurs serront -1. Lorsqu'une valeur "-1" est retournée, cela signifie soit que le temps demandé est top ancien, soit que la valeur n'est plus dans le tableau d'historique interne (c'est-à-dire que la limite des 200 éléments a été atteinte et que les valeurs plus anciennes ont été supprimées).

Client/serveur et composants

Cette commande retourne des informations relatives à l'utilisation de la base de données. Cela signifie qu'elle ne retourne un objet valide contenant des valeurs significatives uniquement lorsqu'elle est appelée :

- 4D en mode local (lorsqu'elle est appelée depuis un composant, elle retourne les données de la base hôte),
- sur le serveur en mode client/serveur.

Si la commande est appelée depuis un 4D distant en mode client/serveur, l'objet est retourné vide.

Dans ce contexte, si vous souhaitez obtenir des informations sur le serveur, le plus simple est de créer une méthode et d'activer l'option "Exécuter sur serveur". Ce principe fonctionne aussi pour un composant : si le composant est utilisé dans un contexte local, la commande retourne des informations sur la base hôte ; dans un contexte de 4D distant, elle retourne des informations sur la base du serveur.

Exemple 1

Vous souhaitez obtenir l'objet "history" dans l'objet retourné :

```
C_OBJET($param)
C_OBJET($measures)
OB FIXER($param;"withHistory";Vrai)
$measures:=Lire mesures base($param)
```

Exemple 2

Vous souhaitez connaître uniquement le nombre global d'octets lus dans le cache ("cacheReadBytes") :

```
C_OBJET($oStats)
C_OBJET($oParams)
OB FIXER($oParams;"path";"DB.cacheReadBytes")
$oStats:=Lire mesures base($oParams)
```

L'objet retourné contiendra, par exemple :

```
{     "DB": {          "cacheReadBytes": {               "value": 9516637          }     } }
```

Exemple 3

Vous souhaitez obtenir les mesures d'octets lus dans le cache au cours des deux dernières minutes :

```
C_OBJET($oParams)
C_OBJET($measures)
OB FIXER($oParams;"path";"DB.cacheReadBytes")
OB FIXER($oParams;"withHistory";Vrai)
OB FIXER($oParams;"historyLength";2*60)
$measures:=Lire mesures base($oParams)
```

Lire parametre base

Lire parametre base ({laTable ;} sélecteur {; valeurAlpha}) -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table du paramètre ou Table par défaut si ce paramètre est omis
sélecteur	Entier long	⇒	Code du paramètre de la base
valeurAlpha	Chaîne	⇐	Valeur alpha du paramètre
Résultat	Réel	⇐	Valeur du paramètre

Description

La commande **Lire parametre base** permet de lire la valeur courante d'un paramètre de la base 4D. Lorsque la valeur du paramètre est une chaîne de caractères, elle est retournée dans le paramètre *valeurAlpha*.

sélecteur désigne le paramètre de la base à lire. 4D vous propose les constantes prédéfinies suivantes, placées dans le thème **Paramètres de la base** :

Constante	Type	Valeur	Comment
Direct2D désactivé	Entier long	0	Voir sélecteur 69 (Direct2D Statut)
Direct2D matériel	Entier long	1	Voir sélecteur 69 (Direct2D Statut)
Direct2D Logiciel	Entier long	3	Voir sélecteur 69 (Direct2D Statut)
Minimum process Web	Entier long	6	**Portée** : 4D local, 4D Server **Conservé entre deux sessions :** Oui **Valeurs possibles** : 0 -> 32 767 **Description** : Nombre minimum de process Web à maintenir en mode sans contexte avec 4D en mode local et 4D Server. Par défaut, la valeur est 0 (cf. ci-dessous).
Maximum process Web	Entier long	7	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 -> 32 767 **Description** : Nombre maximum de process Web à maintenir en mode sans contexte avec 4D en mode local et 4D Server. Par défaut, la valeur est 10. Afin que le serveur Web soit réactif, en mode sans contexte, 4D maintient endormis les process Web pendant 5 secondes, et les réutilise pour traiter les éventuelles requêtes HTTP suivantes. En termes de performances, ce principe est en effet plus avantageux que la création d'un nouveau process à chaque requête. Une fois un process Web réutilisé, il est à nouveau endormi pour 5 secondes, sauf si le nombre maximum de process Web est dépassé (auquel cas il est tué). Si aucune requête n'a été attribuée à un process Web durant les 5 secondes, il est tué, sauf si un nombre minimum de process Web a été fixé et est atteint (auquel cas il est à nouveau endormi). Ces paramètres vous permettent d'ajuster le fonctionnement de votre serveur Web en fonction du nombre de requêtes, de la mémoire disponible, etc.
_o_Mode conversion Web	Entier long	8	**** *Sélecteur inactivé* ****
_o_Taille cache données	Entier long	9	**Portée** : Application 4D **Conservé entre deux sessions** : - **Description** : *Constante obsolète (conservée uniquement pour des raisons de compatibilité)*. L'utilisation de la commande **Lire taille cache** est désormais recommandée.
Appels système 4D mode local	Entier long	10	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Description** : voir sélecteur 12
Appels système 4D Server	Entier long	11	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Description** : voir sélecteur 12
Appels système 4D mode distant	Entier long	12	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Valeurs possibles** : pour les sélecteurs 10, 11 et 12, le paramètre *valeur* est de la forme hexadécimale *0x00aabbcc* dans laquelle : *aa* = nombre minimum de ticks par appel au système (de 0 à 100 inclus) *bb* = nombre maximum de ticks par appel au système (de 0 à 100 inclus) *cc* = nombre de ticks entre deux appels au système (de 0 à 20 inclus). Si une des valeurs est hors de son intervalle, 4D la fixe à son maximum. Vous pouvez également passer dans *valeur* une des trois valeurs suivantes, correspondant à un ensemble de réglages standard : • *valeur* = -1 : priorité maximum allouée à 4D, • *valeur* = -2 : priorité moyenne allouée à 4D, • *valeur* = -3 : priorité minimum allouée à 4D. **Description** : Ce sélecteur vous permet de fixer dynamiquement les réglages des appels système de 4D. En fonction du Sélecteur, le gestionnaire d'appels système sera défini pour : • 4D mode local lorsque la commande est appelée depuis 4D monoposte (*sélecteur=10*). • 4D Server lorsque la commande est appelée depuis 4D Server (*sélecteur=11*). • 4D mode distant lorsque la commande est appelée depuis 4D connecté à 4D Server (*sélecteur=12*). **Note :** Le fonctionnement du sélecteur 12 (Appels système 4D mode distant) diffère suivant que la commande

Constante	Type	Valeur	Comment
Timeout 4D Server	Entier long	13	**FIXER PARAMETRE BASE** est exécutée sur le poste serveur ou sur un poste client : - si la commande est exécutée sur le poste serveur, la nouvelle valeur sera appliquée à tous les postes clients se connectant ultérieurement. - si la commande est exécutée sur un poste client, la nouvelle valeur est appliquée immédiatement au poste client ainsi qu'à tous les postes clients se connectant par la suite au serveur. Vous pouvez utiliser ce fonctionnement pour mettre en place une gestion dynamique et individualisée de la priorité pour chaque poste client. Le principe consiste à exécuter la commande une première fois sur le poste client à configurer puis une seconde fois sur le poste serveur avec la valeur par défaut, qui sera alors utilisée pour les postes client se connectant par la suite. Ce fonctionnement est effectif dans 4D à partir des versions 6.8.6, 2003.3 et 2004. **Attention** : Paramétrer ces sélecteurs de façon inappropriée peut conduire à une forte dégradation des performances de l'application. Il est conseillé de ne modifier les valeurs par défaut qu'en parfaite connaissance de cause. **Portée** : Application 4D si *valeur* positive **Conservé entre deux sessions** : Oui si *valeur* positive **Valeurs possibles** : 0 -> 32 767 **Description** : Valeur du délai avant déconnexion (timeout) accordé par 4D Server aux postes clients. Par défaut, cette valeur est définie dans la page “Client-Serveur/Options réseau” des Propriétés de la base, sur le poste serveur. Le timeout serveur définit la période maximale de non-réponse du client "autorisée", par exemple s'il effectue une opération bloquante. A l'issue de cette période, 4D Server déconnecte le client. Le sélecteur Timeout 4D Server vous permet de fixer un nouveau timeout, exprimé en minutes. Cette possibilité permet en particulier d’augmenter la valeur du timeout avant l’exécution sur le poste client d’une opération bloquante de longue durée, risquant d’entraîner une déconnexion ; par exemple, l’impression d’un grand nombre de pages. Vous disposez en outre de deux possibilités : • effectuer une modification globale et permanente : la nouvelle valeur s’applique à tous les process et est stockée dans les préférences de l’application (équivaut à une modification de la valeur dans la boîte de dialogue des Préférences). Pour cela, passez une valeur **positive** dans le paramètre *valeur*. • effectuer une modification restreinte et temporaire : la nouvelle valeur ne s’applique qu’au process appelant (les autres process conservant la valeur d’origine), et est abandonnée dès que le serveur reçoit un signe d’activité du poste client — par exemple, dès que l’opération est terminée. Cette possibilité est utile pour gérer les opérations longues initiées par des plug-ins. Pour cela, passez une valeur **négative** dans le paramètre *valeur*. Pour définir une connexion “Ouverte en permanence”, passez 0 dans *valeur*. Reportez-vous à l’exemple 1.
Timeout 4D mode distant	Entier long	14	**Portée** (ancienne couche réseau uniquement) : Application 4D si *valeur* positive **Conservé entre deux sessions** : Oui si *valeur* positive **Description** : A utiliser dans des cas très spécifiques. Valeur du délai avant déconnexion (timeout) accordé par le poste 4D distant au poste 4D Server. Par défaut, cette valeur est définie dans la page “Client-Serveur/Options réseau” des Propriétés de la base, sur le poste distant. Le sélecteur Timeout 4D mode distant n'est pris en compte que si vous utilisez l'ancienne couche réseau. Avec la couche *ServerNet* activée, il est ignoré : ce paramétrage est entièrement géré par le sélecteur Timeout 4D Server (13).
Numéro du port	Entier long	15	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Non **Description** : Numéro du port TCP utilisé par le serveur Web 4D avec 4D en mode local et 4D Server. Par défaut, la valeur est 80. Le numéro de port TCP est défini dans la page “Web/Configuration” de la boîte de dialogue des Propriétés de la base. Vous pouvez utiliser les constantes du thème **Numéros de port TCP** pour le paramètre *valeur*. Le sélecteur Numéro du port est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d’accès au mode Développement). Pour plus d’informations sur le numéro de port TCP, reportez-vous à la section **Paramétrages du serveur Web**.
Adresse IP d'écoute	Entier long	16	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Jeu de caractères	Entier long	17	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Process Web simultanés maxi	Entier long	18	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Client minimum process Web	Entier long	19	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 6 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client maximum process Web	Entier long	20	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 7 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.

Constante	Type	Valeur	Commentaire
			Portée : Tous postes 4D distants
Client taille max requêtes Web	Entier long	21	**Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 27 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
			Portée : Tous postes 4D distants
Client numéro de port	Entier long	22	**Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 15 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client adresse IP d'écoute	Entier long	23	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 16 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Propriétés de la base 4D en mode distant.
Client jeu de caractères	Entier long	24	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 17 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client proc Web simultanés maxi	Entier long	25	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 18 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Taille maximum requêtes Web	Entier long	27	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Enreg requêtes 4D Server	Entier long	28	**Portée** : *4D Server, 4D distant* **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 ou de 1 à N (0 = ne pas enregistrer, 1 à N = numéro séquentiel, accolé au nom du fichier). **Description** : Démarrage ou arrêt de l'enregistrement des requêtes standard reçues par 4D Server (hors requêtes Web). Par défaut, la valeur est 0 (pas d'enregistrement de requêtes). 4D Server vous permet d'enregistrer dans un fichier d'historique chaque requête reçue par le poste serveur. Lorsque ce mécanisme est activé, deux fichiers sont créés dans le dossier Logs de la base, à côté du fichier de structure. Il sont nommés 4DRequestsLog_N.txt et 4DRequestsLog_ProcessInfo_N.txt où N est le numéro séquentiel de l'historique. Une fois qu'un fichier atteint une taille de 10 Mo, il est refermé et un nouveau fichier est généré, avec un numéro séquentiel incrémenté. Si un fichier du même nom existe déjà, il est directement remplacé. Vous pouvez définir le numéro de départ de la séquence à l'aide du paramètre *valeur*. Ces fichiers texte stockent dans un format tabulé simple diverses informations concernant chaque requête : heure, numéro de process, utilisateur, taille de la requête, durée de traitement, etc. Ces informations sont particulièrement utiles en phase de mise au point de l'application ou à des fins statistiques. Elles peuvent être importées par exemple dans un logiciel tableur afin d'être traitées.
_o_Enreg requêtes Web	Entier long	29	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Client enreg requêtes Web	Entier long	30	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 = Ne pas enregistrer (défaut), 1 = Enregistrer au format CLF, 2 = Enregistrer au format DLF, 3 = Enregistrer au format ELF, 4 = Enregistrer au format WLF. **Description** : Démarrage ou arrêt de l'enregistrement des requêtes Web reçues par les serveurs Web de tous les postes clients. Par défaut, la valeur est 0 (pas d'enregistrement des requêtes). Le fonctionnement de ce sélecteur est identique à celui du sélecteur 29 ; il s'applique toutefois à tous les postes 4D clients utilisés en tant que serveurs Web. Le fichier "logweb.txt" est dans ce cas automatiquement placé dans le sous-dossier Logs du dossier base 4D client (dossier de cache). Si vous souhaitez définir des valeurs pour certains postes clients uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Numéro automatique table	Entier long	31	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Valeurs possibles** : Toute valeur de type entier long. **Description** : Ce sélecteur permet de modifier ou de lire le numéro unique courant des enregistrements de la table passée en paramètre. "Numéro courant" signifie "dernier numéro utilisé" : si vous modifiez cette valeur à l'aide de **FIXER PARAMETRE BASE**, le prochain enregistrement sera créé avec comme numéro la valeur passée + 1. Ce nouveau numéro est, lui, retourné par la commande **Numerotation automatique** ainsi que dans tout champ de la table auquel la propriété "Incrémentation auto" a été affectée en Structure ou via le SQL. Par défaut, le numéro unique est défini par 4D et correspond à l'ordre de création des enregistrements. Pour des informations supplémentaires, reportez-vous à la documentation de la commande **Numerotation automatique**.
_o_Précision	Entier		

Constante	Type	Valeur	Commentaire
affichage réels	long	32	**** *Sélecteur inactivé* ****
Enreg événements debogage	Entier long	34	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Description** : Démarrage ou arrêt de l'enregistrement séquentiel des événements de programmation de 4D dans le fichier *4DDebugLog*, automatiquement placé dans le sous-dossier Logs de la base, à côté du fichier de structure. Un nouveau format texte tabulé, plus compact, est utilisable pour le fichier d'enregistrement des événements "4DDebugLog[_n].txt" à compter de 4D v14 (_n est le numéro de segment du fichier). **Valeurs possibles** : Entier long contenant un champ de bits (bit field) : valeur = bit1(1)+bit2(2)+bit3(4)+bit4(8)+…). - Le bit 1 (valeur 1) permet de demander à activer le fichier (à noter que toute autre valeur non nulle l'activera également) - Le bit 2 (valeur 2) permet de demander les paramètres d'appel aux méthodes et commandes. - Le bit 3 (valeur 4) permet d'activer le nouveau format tabulé. - Le bit 4 (valeur 8) permet de désactiver l'écriture immédiate de chaque opération sur disque (activée par défaut). L'écriture immédiate est moins rapide mais plus efficace par exemple pour rechercher les causes d'un plantage. Si vous désactivez ce mode, le contenu fichier sera plus compact et il sera généré plus rapidement. - Le bit 5 (valeur 16) permet de désactiver l'enregistrement des appels de plug-ins (activé par défaut). Dans le format non tabulé (ancien format), les durées d'exécution sont exprimées en millisecondes, la valeur "< ms" est affichée si une opération s'exécute en moins d'une milliseconde. Dans le nouveau format tabulé, les durées d'exécution sont exprimées en microsecondes. Exemples : FIXER PARAMETRE BASE(34;1) // active le fichier mode v13 sans les paramètres, avec les durées FIXER PARAMETRE BASE(34;2) // active le fichier mode v13 avec les paramètres et les durées FIXER PARAMETRE BASE(34;2+4) // active le fichier au format v14 avec les paramètres et les durées FIXER PARAMETRE BASE(34;0) // désactive le fichier Afin d'éviter que le fichier n'enregistre une trop grande quantité d'informations, vous pouvez restreindre les commandes 4D à examiner à l'aide du sélecteur 80, Liste commandes enreg. L'option peut être activée dans tout type d'application 4D (4D tous modes, 4D Server, 4D Volume Desktop), en interprété ou en compilé. **Note :** Cette option est proposée uniquement à des fins de débogage, elle ne doit pas être utilisée en production car elle peut entraîner une dégradation des performances de l'application ainsi que la saturation du disque dur. Pour plus d'informations sur le format et l'exploitation du fichier 4DDebugLog[_n].txt, veuillez contacter les services techniques de 4D SAS.
Numéro du port client serveur	Entier long	35	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 à 65535 **Description** : Numéro de port TCP sur lequel 4D Server publie la base de données (à destination des postes 4D distants). Par défaut, la valeur est 19813. La personnalisation de cette valeur permet d'utiliser plusieurs applications 4D client-serveur sur la même machine avec le protocole TCP ; dans ce cas, vous devez spécifier un numéro de port différent pour chaque application. La valeur est stockée dans le fichier de structure de la base. Elle peut être définie avec 4D en mode local mais n'est prise en compte qu'en configuration client-serveur. Lorsque vous modifiez cette valeur, il est nécessaire de redémarrer le poste serveur afin que la nouvelle valeur soit prise en compte.
Inversion des objets	Entier long	37	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0, 1 ou 2 (0 = mode désactivé, 1 = mode automatique, 2 = mode activé). **Description** : Configuration du mode "inversion des objets". Ce mode permet d'inverser en mode Application les formulaires, objets, menus, etc. lorsque la base est affichée sous Windows dans une langue de droite à gauche. Ce mode peut également être configuré dans la page Interface/langue droite à gauche des Propriétés de la base. • La valeur 0 indique que le mode n'est jamais activé, quelle que soit la configuration du système (correspond à la valeur Jamais dans les Propriétés de la base). • La valeur 1 indique que le mode est activé ou non en fonction de la configuration du système (correspond à la valeur Automatique dans les Propriétés de la base). • La valeur 2 indique que le mode est activé, quelle que soit la configuration du système (correspond à la valeur Toujours dans les Propriétés de la base). Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.
Numéro de port HTTPS	Entier long	39	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Client numéro de port HTTPS	Entier long	40	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 à 65535 **Description** : Numéro du port TCP utilisé par les serveurs Web des postes clients pour les connexions sécurisées via SSL (protocole HTTPS). Par défaut, la valeur est 443 (valeur standard). Le fonctionnement de ce sélecteur est identique à celui du sélecteur 39 ; il s'applique toutefois à tous les postes 4D distants utilisés en tant que serveurs Web. Si vous souhaitez modifier la valeur de certains postes clients uniquement, utilisez la boîte de dialogue des Préférences de 4D distant.
Mode Unicode	Entier long	41	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (mode compatibilité) ou 1 (mode Unicode) **Description** : Mode d'exécution courant de la base, relatif au jeu de caractères. 4D prend en charge le jeu de caractères Unicode mais peut fonctionner en mode "compatibilité" (basé sur le jeu de caractères ASCII Mac). Par défaut, les bases de données converties sont exécutées en mode compatibilité (0) et les bases créées à partir de la version 11 sont exécutées en mode Unicode. Le mode d'exécution est contrôlé via une option des préférences et peut également être lu ou (à des fins de test) modifié via ce sélecteur. La modification de cette option nécessite le redémarrage de la base pour être prise en compte. A noter que, dans le cadre d'un composant, il n'est pas possible de modifier cette valeur, mais uniquement de la lire.
			Portée : Base de données **Conservé entre deux sessions** : Oui

Constante	Type	Valeur	Commentaire
SQL autocommit	Entier long	43	**Valeurs possibles** : 0 (désactivation) ou 1 (activation) **Description** : Activation ou désactivation du mode SQL auto-commit. Par défaut, la valeur est 0 (mode désactivé) Le mode auto-commit permet de renforcer l'intégrité référentielle de la base. Lorsque ce mode est actif, les requêtes **SELECT**, **INSERT**, **UPDATE**, **DELETE** (SIUD) sont automatiquement incluses dans des transactions lorsqu'elles sont exécutées en-dehors de toute transaction. Ce mode peut également être défini dans les préférences de la base.
Casse caractères moteur SQL	Entier long	44	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (casse non prise en compte) ou 1 (casse prise en compte) **Description** : Activation ou désactivation de la prise en compte de la casse des caractères pour les comparaisons de chaînes effectuées par le moteur SQL. Par défaut, la valeur est 1 (casse prise en compte) : le moteur SQL établit une différence entre les majuscules et les minuscules ainsi qu'entre les caractères accentués lors des comparaisons de chaînes (tris et recherches). Par exemple "ABC" = "ABC" mais "ABC" # "Abc" et "abc" # "âbc" . Dans certains cas, par exemple pour aligner le fonctionnement du moteur SQL sur celui du moteur 4D, vous pourrez souhaiter que les comparaisons de chaînes ne tiennent pas compte de la casse ("ABC"="Abc"="âbc"). Cette option peut également être définie dans la **Page SQL** des Propriétés de la base.
Enreg requêtes client	Entier long	45	**Portée** : Poste 4D distant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 ou de 1 à N (0 = ne pas enregistrer, 1 à N = numéro séquentiel, accolé au nom du fichier). **Description** : Démarrage ou arrêt de l'enregistrement des requêtes standard effectuées par le poste client 4D ayant exécuté la commande (hors requêtes Web). Par défaut, la valeur est 0 (pas d'enregistrement des requêtes). 4D vous permet d'enregistrer l'historique des requêtes effectuées par le poste client. Lorsque ce mécanisme est activé, deux fichiers sont créés sur le poste client, dans le sous-dossier Logs du dossier local de la base. Il sont nommés 4DRequestsLog_N.txt et 4DRequestsLog_ProcessInfo_N.txt où N est le numéro séquentiel de l'historique. Une fois que le fichier 4DRequestsLog atteint une taille de 10 Mo, il est refermé et un nouveau fichier est généré, avec un numéro séquentiel incrémenté. Si un fichier du même nom existe déjà, il est directement remplacé. Vous pouvez définir le numéro de départ de la séquence à l'aide du paramètre valeur. Ces fichiers texte stockent dans un format tabulé simple diverses informations concernant chaque requête : heure, numéro de process, taille de la requête, durée de traitement, etc. Ces informations sont particulièrement utiles en phase de mise au point de l'application ou à des fins statistiques.
Chercher par formule serveur	Entier long	46	**Portée** : Table et process courants **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (utiliser le paramétrage de la base), 1 (exécuter sur le client) ou 2 (exécuter sur le serveur) **Description** : Emplacement de l'exécution des commandes **CHERCHER PAR FORMULE** et **CHERCHER PAR FORMULE DANS SELECTION** pour la *table* passée en paramètre. Dans le cadre de l'exploitation d'une base en client-serveur, les commandes de recherche "par formule" peuvent exécutées soit sur le serveur soit sur le client : • dans les bases de données créées à partir de 4D v11 SQL, ces commandes sont exécutées sur le serveur. • dans les bases de données converties, ces commandes sont exécutées sur le client, comme dans les versions précédentes de 4D. • dans les bases de données converties, une préférence spécifique permet de modifier globalement le lieu d'exécution de ces commandes. Cette différence de lieu d'exécution influe sur les performances de l'application (l'exécution sur le serveur est généralement plus rapide) mais également sur la programmation. En effet, la valeur des composantes de la formule (notamment les variables appelées via une méthode) diffère suivant le contexte d'exécution. Vous pouvez utiliser ce sélecteur pour adapter ponctuellement le fonctionnement de votre application. Si vous passez 0 dans le paramètre *valeur*, l'emplacement d'exécution des commandes de recherche "par formule" dépendra de la configuration de la base : dans les bases créées avec 4D v11 SQL, les commandes seront exécutées sur le serveur. Dans les bases converties, elles seront exécutées sur le client ou le serveur en fonction des préférences de la base. Passez 1 ou 2 dans *valeur* pour "forcer" l'exécution des commandes respectivement sur le client ou sur le serveur. Reportez-vous à l'exemple 4. **Note :** Si vous souhaitez pouvoir activer les jointures "type SQL" (cf. sélecteur Jointures chercher par formule), vous devez toujours exécuter les formules sur le serveur afin qu'elle ait accès aux enregistrements. Attention, dans ce contexte, la formule ne doit pas contenir d'appel à une méthode, sinon elle est automatiquement basculée sur le poste distant.
Trier par formule serveur	Entier long	47	**Portée** : Table et process courants **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (utiliser le paramétrage de la base), 1 (exécuter sur le client) ou 2 (exécuter sur le serveur) **Description** : Emplacement de l'exécution de la commande **TRIER PAR FORMULE** pour la table passée en paramètre. Dans le cadre de l'exploitation d'une base en client-serveur, la commande **TRIER PAR FORMULE** peut être exécutée soit sur le serveur soit sur le client. Ce sélecteur permet de définir l'emplacement de l'exécution de cette commande (serveur ou client). Ce mode peut également être défini dans les préférences de la base. Pour plus d'informations, reportez-vous à la description du sélecteur 46, Chercher par formule serveur. **Note :** Si vous souhaitez pouvoir activer les jointures "type SQL" (cf. sélecteur Jointures chercher par formule), vous devez toujours exécuter les formules sur le serveur afin qu'elle ait accès aux enregistrements. Attention, dans ce contexte, la formule ne doit pas contenir d'appel à une méthode, sinon elle est automatiquement basculée sur le poste distant.
Synchro auto dossier	Entier long	48	**Portée** : Poste 4D distant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (pas de synchronisation), 1 (synchronisation auto) ou 2 (demander). **Description** : Mode de synchronisation dynamique du dossier *Resources* du poste client 4D ayant exécuté la commande avec celui du serveur. Lorsque le contenu du dossier *Resources* sur le serveur a été modifié ou qu'une demande de synchronisation a été émise (via l'explorateur de ressources ou suite à l'exécution de la commande **NOTIFIER MODIFICATION DOSSIER RESOURCES**), le serveur notifie les clients connectés. Trois modes de synchronisation sont alors possibles côté client. Le sélecteur Synchro auto dossier Resources vous

Ressource Constante	Type	Valeur	Comment Permet de définir le mode à utiliser pour le poste client et la session courante :
Jointures chercher par formule	Entier long	49	• 0 (valeur par défaut) : pas de synchronisation dynamique (la demande de synchronisation est ignorée) • 1 : synchronisation dynamique automatique • 2 : affichage d'une boîte de dialogue sur les postes clients, avec possibilité d'effectuer ou de refuser la synchronisation. Le mode de synchronisation peut également être défini globalement dans les Propriétés de la base. **Portée** : Process courant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (utiliser paramétrages de la base), 1 (toujours utiliser les liens auto) ou 2 (utiliser les jointures SQL si possible). **Description** : Mode de fonctionnement des commandes **CHERCHER PAR FORMULE** et **CHERCHER PAR FORMULE DANS SELECTION** relatif à l'utilisation de "jointures SQL". Dans les bases de données créées à compter de la version 11.2 de 4D v11 SQL, ces commandes effectuent des jointures sur le modèle des jointures SQL. Ce mécanisme permet de modifier la sélection d'une table en fonction d'une recherche effectuée sur une autre table sans que les tables soient reliées par un lien automatique (condition nécessaire dans les versions précédentes de 4D). Le sélecteur Jointures chercher par formule vous permet de définir le mode de fonctionnement des commandes de recherche par formule pour le process courant : • 0 : utiliser les paramètres courants de la base (valeur par défaut). Dans les bases de données créées à compter de la version 11.2 de 4D v11 SQL, les "jointures SQL" sont toujours activées pour les recherches par formule. Dans les bases de données converties, ce mécanisme est inactivé par défaut pour des raisons des compatibilité mais peut être mis en oeuvre via une préférence. • 1 : toujours utiliser les liens auto (= fonctionnement des versions précédentes de 4D). Dans ce mode, un lien est nécessaire pour définir la sélection d'une table en fonction de recherches effectuées dans une autre table. 4D n'effectue pas de "jointures SQL". • 2 : utiliser les jointures SQL si possible (= fonctionnement par défaut des bases créées en version 11.2 et suivantes de 4D v11 SQL). Dans ce mode, 4D établit des "jointures SQL" pour les recherches par formule lorsque la formule s'y prête (à deux exceptions près, voir la description de la commande commandes **CHERCHER PAR FORMULE** ou **CHERCHER PAR FORMULE DANS SELECTION**. **Note :** Avec 4D en mode distant, les "jointures SQL" ne peuvent être utilisées que si les formules sont exécutées sur le serveur (elles doivent avoir accès aux enregistrements). Pour configurer le lieu d'exécution des formules, reportez-vous aux sélecteurs 46 et 47.
Niveau de compression HTTP	Entier long	50	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Seuil de compression HTTP	Entier long	51	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Taille pile process base server	Entier long	53	**Portée** : 4D Server **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif. **Description** : Taille de la pile allouée à chaque process système préemptif sur le serveur, exprimée en octets. La taille par défaut est déterminée par le système. Les process système préemptifs (process de type Process base 4D client) sont chargés de contrôler les process clients 4D principaux. La taille allouée par défaut à la pile de chaque process préemptif permet un bon confort d'exécution mais peut s'avérer conséquente lorsque de très nombreux process (plusieurs centaines) sont créés. A des fins d'optimisation, cette taille peut être diminuée sensiblement si les opérations effectuées par la base s'y prêtent (par exemple si la base n'effectue pas de tris sur de grosses quantités d'enregistrements). Des valeurs de 512 voire de 256 Ko sont possibles. Attention, le sous-dimensionnement de la pile est critique et peut nuire au fonctionnement de 4D Server. Le réglage de ce paramètre est à effectuer avec précaution et doit tenir compte des conditions d'utilisation de la base (nombre d'enregistrements, types d'opérations, etc.). Pour être pris en compte, ce paramétrage doit être exécuté sur le poste serveur (par exemple dans la méthode base Sur démarrage serveur).
Timeout connexions inactives	Entier long	54	**Portée** : Application 4D sauf si valeur négative **Conservé entre deux sessions** : Non **Valeurs possibles** : Valeur entière exprimant une durée en secondes. La valeur peut être positive (nouvelles connexions) ou négative (connexions existantes). Par défaut, la valeur est 20. **Description** : Délai maximum d'inactivité (timeout) des connexions au moteur de base de données et au serveur SQL de 4D ainsi que, en mode *ServerNet* (nouvelle couche réseau), au serveur d'applications 4D. Lorsqu'une connexion inactive atteint ce délai, elle est automatiquement mise en veille, ce qui se traduit par le gel de la session client/serveur et la fermeture du socket réseau. Dans la fenêtre d'administration du serveur, le process utilisateur prend l'état "Postponed". Ce fonctionnement est entièrement transparent pour l'utilisateur : dès qu'il y a reprise d'activité sur la connexion mise en veille, le socket est automatiquement rouvert et la session client/serveur restaurée. Ce paramétrage permet, d'une part, d'économiser des ressources sur le serveur : les connexions mises en veille referment le socket et libèrent un processus sur le serveur. D'autre part, il permet d'éviter les pertes de connexions dues aux fermetures par les pare-feux des sockets inactifs. La valeur de timeout des connexions inactives doit pour cela être inférieure à celle du pare-feu. Si vous passez une valeur positive dans *valeur*, elle s'applique à toutes les nouvelles connexions dans tous les process. Si vous passez une valeur négative, elle s'applique aux connexions ouvertes dans le process courant. Si vous passez 0, les connexions inactives ne sont pas soumises à un timeout. Ce paramètre peut être défini côté serveur et côté client. Si vous passez deux durées différentes, la plus courte sera prise en compte. Généralement, vous n'aurez pas besoin de modifier cette valeur.
PHP adresse	Entier		**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Chaîne formatée du type "nnn.nnn.nnn.nnn" (par exemple "127.0.0.1").

Constante	Type	Valeur	Commentaire
IP interpréteur	Texte	55	**Description** : Adresse IP utilisée localement par 4D pour communiquer avec l'interpéteur PHP via fastcgi. Par défaut, la valeur est "127.0.0.1". Cette adresse doit correspondre à la machine sur laquelle se trouve 4D. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations sur l'interpréteur PHP de 4D, reportez-vous au manuel *Mode Développement*.
PHP port interpréteur	Entier long	56	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Valeur de type entier long positif. Par défaut, la valeur est 8002. **Description** : Numéro du port TCP utilisé par l'interpréteur PHP de 4D. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations sur l'interpréteur PHP de 4D, reportez-vous au manuel *Mode Développement*.
PHP nombre enfants	Entier long	57	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Valeur de type entier long positif. Par défaut, la valeur est 5. **Description** : Nombre de process enfants à créer et à maintenir localement par l'interpréteur PHP de 4D. Pour des raisons d'optimisation, l'interpréteur PHP crée et utilise un ensemble (pool) de process système appelés "process enfants" pour traiter les demandes d'exécution de scripts. Vous pouvez faire varier le nombre de process enfants en fonction des besoins de votre application. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations sur l'interpréteur PHP de 4D, reportez-vous au manuel *Mode Développement*. **Note :** Sous Mac OS, tous les process enfants partagent le même port. Sous Windows, chaque process enfant utilise un numéro de port spécifique. Le premier numéro est celui défini pour l'interpréteur PHP, les autres process enfants l'incrémentent. Par exemple, si le port par défaut est le 8002 et que vous lancez 5 process enfants, ils utiliseront les ports 8002 à 8006.
PHP nombre requêtes max	Entier long	58	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Valeur de type entier long positif. Par défaut, la valeur est 500. **Description** : Nombre maximum de requêtes acceptées par l'interpréteur PHP. Lorsque ce nombre maximum est atteint, l'interpréteur retourne des erreurs du type "serveur occupé". Pour des raisons de sécurité ou de performance, vous pouvez modifier cette valeur. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations, reportez-vous à la documentation de fastcgi-php. **Note :** Côté 4D, ces paramètres sont appliqués dynamiquement, il n'est pas nécessaire de quitter le programme pour leur prise en compte. En revanche, si l'interpréteur PHP était déjà lancé, il est nécessaire de le quitter et de le relancer pour qu'il prenne en compte ces modifications.
PHP utiliser interpréteur externe	Entier long	60	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : 0 = utiliser interpréteur interne, 1 = utiliser interpréteur externe **Description** : Valeur indiquant si les requêtes PHP de 4D sont adressées à l'interpréteur interne fourni par 4D ou un interpréteur externe. Par défaut, la valeur est 0 (utilisation de l'interpréteur fourni par 4D). Si vous souhaitez utiliser votre propre interpréteur PHP, par exemple pour bénéficier de modules supplémentaires ou d'une configuration spécifique, passez 1 dans *valeur*. Dans ce cas, 4D ne démarrera pas son interpréteur en cas de requête PHP. L'interpréteur PHP personnalisé doit avoir été compilé en fastcgi et se trouver sur la même machine que le moteur 4D. A noter que dans ce cas, vous devez entièrement gérer l'interpréteur, il n'est ni démarré ni stoppé par 4D. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base.
Taille maxi mémoire temporaire	Entier long	61	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif. **Description** : Taille maximum de mémoire temporaire que 4D pourra allouer à chaque process, exprimée en Mo. Par défaut, la valeur est 0 (pas de taille maximum). 4D utilise une mémoire temporaire spéciale dédiée aux opérations d'indexation et de tri. Cette mémoire a pour but de préserver la mémoire cache "classique" lors d'opérations massives. Elle n'est activée qu'en cas de besoin. Par défaut, la taille de la mémoire temporaire n'est limitée que par les ressources disponibles (en fonction de la configuration mémoire du système). Ce mécanisme convient à la plupart des applications. Toutefois, dans certains contextes spécifiques, notamment lorsqu'une application client-serveur effectue simultanément un grand nombre de tris séquentiels, la taille de la mémoire temporaire peut augmenter de façon critique, jusqu'à rendre le système instable. Dans ce contexte, fixer une taille maximum à la mémoire temporaire permet de préserver le fonctionnement de l'application. En contrepartie, la vitesse d'exécution pourra être affectée : lorsque la taille maximum est atteinte pour un process, 4D utilise des fichiers disque, ce qui peut ralentir les traitements. Pour des besoins tels que ceux décrits ci-dessus, une taille maximum d'environ 50 Mo est généralement un bon compromis. La valeur idéale sera cependant à déterminer en fonction des spécificités de l'application et résultera généralement de tests en volumétrie réelle.
Liste de chiffrement SSL	Chaîne	64	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Suite de chaînes séparées par des deux-points **Description :** Liste de chiffrement (*cipher list*) utilisée par 4D pour le protocole sécurisé. Cette liste permet de modifier la priorité des algorithmes de chiffrement mis en oeuvre par 4D. Par exemple, vous pouvez passer la chaîne suivante dans le paramètre *valeur* : "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". Pour une description complète de la syntaxe de la liste de chiffrement, reportez-vous à la page ciphers sur le site de OpenSSL. Ce paramétrage est global à l'application 4D (il concerne le serveur HTTP, le serveur SQL, les connexions client/serveur ainsi que le client HTTP et les commandes 4D faisant appel au protocole sécurisé). Lorsque la liste de chiffrement a été modifiée, vous devez redémarrer le serveur concerné pour que le nouveau paramétrage soit pris en compte. Pour réinitialiser la liste de chiffrement à sa valeur par défaut (stockée en dur dans le fichier SLI), appelez la commande **FIXER PARAMETRE BASE** et passez une chaîne vide ("") dans le paramètre *valeur*. **Note :** Avec la commande **Lire parametre base**, la liste de chiffrement est retournée dans le paramètre optionnel *valeurAlpha* et le paramètre de retour vaut toujours 0.
Taille minimum libération	Entier	66	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif > 1. **Description** : Taille minimum de mémoire à libérer du cache de la base de données lorsque le moteur a besoin d'y faire de la place pour y allouer un objet (valeur en octets).

Constante	Type	Valeur	Commentaire
Cache lecteur	Entier long	68	Ce sélecteur a pour but de permettre de réduire le nombre de libérations de données du cache afin d'obtenir des gains de performances. Vous pouvez faire varier ce paramétrage en fonction de la taille du cache et de celle des blocs de données manipulées dans votre base. Par défaut, si ce sélecteur n'est pas utilisé, 4D décharge au minimum 10 % du cache en cas de besoin de place. **Portée**: Application 4D **Conservé entre deux sessions** : Non
Direct2D statut	Entier long	69	**Description** : Mode d'activation de l'implémentation de Direct2D sous Windows. **Valeurs possibles** : Une des constantes suivantes (mode 3 par défaut) : Direct2D désactivé (0) : le mode Direct2D n'est pas activé, la base fonctionne dans le mode précédent (GDI/GDIPlus). Direct2D matériel (1) : utilisation de Direct2D en contexte graphique matériel dans toute l'application 4D. Si ce contexte n'est pas disponible, utilisation du contexte graphique Direct2D logiciel (hormis sous Vista, pour des raisons de performances dans ce cas le mode GDI/GDIPlus est utilisé). Direct2D Logiciel (3) (Mode par défaut) : à partir de Windows 7, utilisation de Direct2D en contexte graphique logiciel dans toute l'application 4D. Sous Vista, pour des raisons de performances le mode GDI/GDIPlus est utilisé. **Note de compatibilité :** A compter de 4D v14, les modes hybrides sont désactivés et sont redirigés vers les modes disponibles (l'ancien mode 2 équivaut au mode 1 ; les anciens modes 4 et 5 sont équivalents au mode 3).
Direct2D lire statut actif	Entier long	74	**Note :** Ce sélecteur peut être utilisé uniquement avec la commande **Lire parametre base**, sa valeur ne peut pas être fixée. **Description** : Retourne l'implémentation active de Direct2D sous Windows. **Valeurs possibles** : 0, 1, 2, 3, 4 ou 5 (cf. valeurs du sélecteur 69). La valeur retournée dépend de la disponibilité de Direct2D, du matériel et de la qualité de la prise en charge de Direct2D par le système d'exploitation. Par exemple, si vous exécutez : `FIXER PARAMETRE BASE(Direct2D Statut;Direct2D Matériel)` `$mode:=Lire parametre base(Direct2D Lire statut actif)` - sur Windows 7 et suivants, *$mode* vaudra 1 si le système détecte un matériel compatible Direct2D, sinon *$mode* vaudra 3 (contexte logiciel). - sur Windows Vista, *$mode* vaudra 1 si le système détecte un matériel compatible Direct2D, sinon *$mode* vaudra 0 (désactivation de Direct2D). - sur Windows XP, *$mode* vaudra toujours 0 (incompatibilité avec Direct2D).
Enreg diagnostic	Entier long	79	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 ou 1 (0 = ne pas enregistrer, 1 = enregistrer) **Description** : Démarrage ou arrêt de l'enregistrement du fichier de diagnostic de 4D. Par défaut, la valeur est 0 (pas d'enregistrement). 4D vous permet d'enregistrer de manière continue dans un fichier de diagnostic un ensemble d'événements relatifs au fonctionnement interne de l'application. Les informations contenues dans ce fichier sont destinées à la mise au point des applications 4D et pourront être analysées avec l'aide des services techniques de 4D. Lorsque vous passez 1 dans ce sélecteur, le fichier de diagnostic, nommé *NomBase*.txt, est automatiquement créé (ou ouvert) dans le dossier **Logs** de la base. Une fois que le fichier atteint une taille de 10 Mo, il est refermé et un nouveau fichier *NomBase*_N.txt est généré, avec un numéro séquentiel N incrémenté. A noter qu'il est possible d'inclure des informations personnalisées dans ce fichier à l'aide de la commande **ENREGISTRER EVENEMENT**.
Liste commandes enreg	Chaîne	80	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Chaîne contenant la liste des numéros des commandes 4D à enregistrer (séparées par des points-virgules), "all" pour enregistrer toutes les commandes ou "" (chaîne vide) pour n'enregistrer aucune commande. **Description** : Liste des commandes 4D à enregistrer dans le fichier de débogage (cf. sélecteur 34, Enreg événements debogage). Par défaut, toutes les commandes 4D sont enregistrées. Ce sélecteur vous permet de restreindre la quantité d'informations stockées dans le fichier de débogage en limitant les commandes 4D dont vous souhaitez enregistrer l'exécution. Par exemple, vous pouvez écrire : `FIXER PARAMETRE BASE(Liste commandes enreg;"277;341") //enregistrer uniquement les commandes CHERCHER et CHERCHER DANS SELECTION`
Correcteur orthographique	Entier long	81	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (défaut) = correcteur OS X natif (Hunspell désactivé), 1 = correcteur Hunspell actif. **Description** : Permet d'activer le correcteur orthographique Hunspell sous OS X. Par défaut, sur cette plate-forme le correcteur natif est activé. Vous pouvez souhaiter utiliser le correcteur Hunspell par exemple pour unifier l'interface de vos applications multiplates-formes (sous Windows, seul le correcteur Hunspell est disponible). Pour plus d'informations, reportez-vous au paragraphe **Prise en charge des dictionnaires Hunspell**.
Prise en charge QuickTime	Entier long	82	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (défaut) = QuickTime désactivé, 1 = QuickTime activé. **Description** : Dans 4D à compter de la v14, par défaut les codecs QuickTime ne sont plus pris en charge. Par compatibilité, vous pouvez les réactiver dans votre base à l'aide de ce sélecteur. La modification de cette option nécessite le redémarrage de la base. A noter toutefois que la prise en charge de QuickTime sera définitivement supprimée dans les prochaines versions de 4D.
JSON fuseau	Entier		**Portée** : Process courant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 = ignorer le fuseau horaire local, 1 (défaut) = tenir compte du fuseau horaire. **Description** : Par défaut, la conversion des dates 4D vers le format JSON tient compte du fuseau horaire local. Par exemple, la conversion de la date !23/08/2013! donne "2013-08-22T22:00:00Z" au format JSON lorsque l'opération est effectuée en France en été (GMT+2). Ce principe est conforme au fonctionnement standard de JavaScript.

Constante	Type	Valeur	Comment
Utiliser ancienne couche réseau	Entier long	87	Ce fonctionnement peut être source d'erreurs si vous souhaitez envoyer des valeurs de date en JSON à une personne qui se trouve dans un autre fuseau horaire. C'est le cas par exemple pour l'exportation d'une table avec **Selection vers JSON** en France destiné à être réimporté aux USA avec **JSON VERS SELECTION**. Par défaut, les dates étant réinterprétées dans chaque fuseau horaire, les valeurs stockées dans la base seront différentes. Dans ce cas, vous pouvez modifier le mode de conversion des dates afin qu'il ne tienne pas compte du fuseau horaire en passant 0 dans ce sélecteur. La conversion de la date !23/08/2013! donnera alors "2013-08-23T00:00:00Z" dans tous les cas. **Portée :** 4D local, 4D Server. **Conservé entre deux sessions :** Oui **Description :** Fixe ou lit le statut courant de l'ancienne couche réseau pour les connexions client/serveur. L'ancienne couche réseau est obsolète à compter de 4D v14 R5 et doit être progressivement remplacée dans vos applications par la couche réseau *ServerNet*. *ServerNet* sera nécessaire dans les prochaines versions de 4D afin de permettre aux applications 4D de tirer parti des futures évolutions réseau. Pour des raisons de compatibilité, l'ancienne couche réseau est toujours prise en charge afin de faciliter la transition des applications existantes (elle reste utilisée par défaut dans les applications converties depuis des versions antérieures à la v14 R5). Passez 1 dans ce paramètre pour utiliser l'ancienne couche réseau (et désactiver *ServerNet*), et passez 0 pour désactiver l'ancienne couche réseau (et utiliser *ServerNet*). Cette propriété peut également être définie à l'aide de l'option "Utiliser l'ancienne couche réseau" présente dans la **Page Compatibilité** des Propriétés de la base (voir section **Options réseau et Client-serveur** ; dans cette section, vous trouverez aussi un paragraphe décrivant la stratégie de migration. Nous vous recommandons d'activer *ServerNet* dès que possible). Il est nécessaire de redémarrer l'application pour que ce paramètre soit pris en compte. Cette option n'est pas disponible dans 4D Server v14 R5 version 64 bits pour OS X, qui prend en charge uniquement *ServetNet* (elle vaut toujours 0). **Valeurs possibles :** 0 ou 1 (0 = ne pas utiliser l'ancienne couche, 1 = utiliser l'ancienne couche) **Valeur par défaut :** 0 dans les applications créées avec 4D v14 R5 ou suivantes, 1 dans les applications converties depuis 4D v14 R4 ou précédentes.
Numéro de port serveur SQL	Entier long	88	**Portée** : 4D mode local et 4D Server. **Conservé entre deux sessions** : Oui **Description** : Permet de lire ou de fixer le numéro du port TCP utilisé par le serveur SQL intégré de 4D en mode local ou de 4D Server. Par défaut, la valeur est 19812. Le numéro de port TCP peut également être défini dans la page "SQL" de la boîte de dialogue des Propriétés de la base. Lorsque ce sélecteur est utilisé en écriture, la propriété de la base est mise à jour. **Valeurs possibles** : 0 à 65535. **Valeur par défaut** : 19812
Limitation nombre journaux	Entier long	90	**Portée** : 4D local, 4D Server. **Conservé entre deux sessions** : Non **Valeurs possibles** : Toute valeur entière, 0 = conserver tous les journaux **Description** : Nombre maximum de fichiers à conserver par roulement pour chaque type de journal. Par défaut, tous les fichiers sont conservés. Si vous passez une valeur N, seuls les N fichiers les plus récents seront conservés, le plus ancien étant automatiquement effacé à la création d'un nouveau. Ce paramétrage s'applique à chacun des fichiers journaux suivants : journal des requêtes (sélecteurs 28 et 45), journal de débogage (sélecteur 34), journal des événements (sélecteur 79), ainsi que l'historique des requêtes Web et l'historique debug des requêtes Web (sélecteurs 29 et 84 de la commande **WEB FIXER OPTION**).
Nombre de formules en cache	Entier long	92	**Portée** : Application 4D. **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif **Valeur par défaut** : 0 (pas de cache) **Description** : Fixe ou lit le nombre maximum de formules à conserver dans le cache des formules, qui est utilisé par la commande **EXECUTER FORMULE**. Cette limite est appliquée à tous les process, mais chaque process dispose de son propre cache de formules. Placer des formules dans le cache accélère l'exécution de la commande **EXECUTER FORMULE** en mode compilé puisque chaque formule en cache est tokenisée une seule fois dans ce cas. Lorsque vous modifiez la valeur du cache, son contenu est réinitialisé même si la nouvelle valeur est supérieure à la précédente. Une fois le nombre maximum de formules en cache atteint, toute nouvelle formule exécutée écrase la plus ancienne dans le cache (mode FIFO). Ce paramètre est pris en compte uniquement dans les bases ou les composants compilés.
Périodicité écriture cache	Entier long	95	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Non **Valeurs possibles** : entier long > 1 (secondes) **Description** : Permet de lire ou de fixer la valeur courante de périodicité de l'écriture du cache de données sur le disque, exprimée en secondes. Si elle est modifiée, cette valeur remplace la valeur définie par l'option **Ecriture cache toutes les <n> secondes/minutes** dans la **Page Base de données/Mémoire** des Propriétés de la base durant la session courante (elle n'est pas stockée dans les Propriétés de la base).

Exemple 1

Cette méthode permet de récupérer les valeurs courantes du minuteur interne de 4D :

```
C_ENTIER LONG($ticksbtwcalls;$maxticks;$minticks;$lparams)
Si(Type application=4D mode local) ` Si nous sommes en 4D local
   $lparams:=Lire parametre base(Appels système 4D mode local)
   $ticksbtwcalls:=$lparams &0x00ff
   $maxticks:=($lparams>>8)&0x00ff
   $minticks:=($lparams>>16)&0x00ff
Fin de si
```

Exemple 2

Le sélecteur 16 (Adresse IP d'écoute) permet d'obtenir l'adresse IP sur laquelle le serveur Web 4D reçoit les requêtes HTTP. L'adresse obtenue est de forme hexadécimale. L'exemple suivant permet de décomposer l'adresse IP reçue :

```
C_ENTIER LONG($a;$b;$c;$d)
C_ENTIER LONG($addr)
$addr:=Lire parametre base(Adresse IP d'écoute)
$a:=($addr>>24)&0x000000ff
$b:=($addr>>16)&0x000000ff
$c:=($addr>>8)&0x000000ff
$d:=$addr&0x000000ff
```

LISTE COMPOSANTS

LISTE COMPOSANTS (tabComposants)

Paramètre	Type		Description
tabComposants	Tableau texte	⇦	Noms des composants

Description

La commande **LISTE COMPOSANTS** dimensionne et remplit le tableau *tabComposants* avec les noms des composants chargés par l'application 4D pour la base hôte courante.

A l'ouverture d'une base, 4D charge les composants valides situés dans le(s) dossier(s) Components :

- le dossier Components situé à côté du fichier de structure (s'il y en a un),
- le dossier Components situé à côté de l'application 4D exécutable.

Rappel : Si un même composant est placé aux deux endroits, 4D charge uniquement celui situé à côté de la structure.

Cette commande peut être appelée depuis la base hôte ou depuis un composant. Si la base n'utilise pas de composant, le tableau *tabComposants* est retourné vide.

Les noms des composants sont les noms des fichiers de structure des bases matrices (.4db, .4dc ou .4dbase). Cette commande permet de mettre en place des architectures et des interfaces modulaires proposant des fonctionnalités supplémentaires en fonction de la présence des composants.

Pour plus d'informations sur les composants 4D, reportez-vous au manuel *Mode Développement*.

LISTE PLUGINS

LISTE PLUGINS (tabNuméros ; tabNoms)

Paramètre	Type		Description
tabNuméros	Tableau entier long	⇐	Numéros des plug-ins
tabNoms	Tableau chaîne	⇐	Noms des plug-ins

Description

La commande **LISTE PLUGINS** remplit les tableaux *tabNuméros* et *tabNoms* avec les numéros et les noms des plug-ins chargés par l'application 4D et utilisables. Les deux tableaux sont automatiquement dimensionnés et synchronisés par la commande.

Note : Vous pouvez comparer les valeurs retournées dans le tableau *tabNuméros* avec les constantes du thème **Licence disponible**.

LISTE PLUGINS prend en compte tous les plug-ins, y compris les plug-ins intégrés (par exemple 4D Chart) et les plug-ins des éditeurs tiers.

Mode compile

Mode compile {(*)} -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Retourner l'information de la base hôte
Résultat	Booléen	⮌	Mode compilé (Vrai), mode interprété (Faux)

Description

La fonction **Mode compile** teste si la base tourne en mode compilé (Vrai) ou en mode interprété (Faux).

Le paramètre facultatif * est utile dans le cadre d'une architecture utilisant des composants : il permet de déterminer la base (hôte ou composant) dont vous souhaitez connaître le mode d'exécution.

- Lorsque la commande est appelée depuis un composant :
  - si le paramètre * est passé, la commande retourne **Vrai** ou **Faux** en fonction du mode d'exécution de la base hôte,
  - si le paramètre * n'est pas passé, la commande retourne **Vrai** ou **Faux** en fonction du mode d'exécution du composant.
- Lorsque la commande est appelée depuis une méthode d'une base hôte, elle retourne toujours **Vrai** ou **Faux** en fonction du mode d'exécution de la base hôte.

Exemple

Dans une de vos méthodes, vous avez placé du code pour déboguer la base lorsque vous êtes en mode interprété. Vous pouvez précéder ce code d'un test qui appelle la fonction **Mode compile** :

```
  ` ...
Si(Non(Mode compile))
  ` Mettre du code pour déboguer votre base ici
Fin de si
  ` ...
```

NOTIFIER MODIFICATION DOSSIER RESOURCES

NOTIFIER MODIFICATION DOSSIER RESOURCES
Ne requiert pas de paramètre

Description

La commande **NOTIFIER MODIFICATION DOSSIER RESOURCES** permet de "forcer" l'envoi par 4D Server d'une notification indiquant à tous les postes 4D connectés que le contenu du dossier **Resources** de la base a été modifié, afin de leur permettre de synchroniser leur dossier **Resources** local. Cette commande permet en particulier de gérer la synchronisation des dossiers **Resources** téléchargés sur les postes distants lorsque le dossier **Resources** de la base est modifié via une procédure stockée sur le serveur.
Pour plus d'informations sur la gestion du dossier **Resources** en mode distant, reportez-vous au *Guide de référence de 4D Server*.

Seule l'information de modification est envoyée par cette commande. Les postes distants réagiront en fonction du paramétrage courant. Les options sont :

- pas de synchronisation du dossier **Resources** local en cours de session,
- synchronisation automatique du dossier **Resources** local en cours de session,
- affichage d'une alerte afin que l'utilisateur effectue une synchronisation s'il le souhaite.

Le paramétrage courant peut être défini soit :

- au niveau global de la base via le paramètre **Mise à jour du dossier Resources en cours de session** des Propriétés de la base. Dans ce cas, il s'applique à tous les postes distants ;
- localement, à l'aide de la commande **FIXER PARAMETRE BASE** exécutée sur le poste distant (sélecteur Synchro auto dossier Resources). Dans ce cas, il "surcharge" celui de la base et s'applique uniquement au poste distant pour la session.

OUVRIR BASE

OUVRIR BASE (cheminFichier)

Paramètre	Type		Description
cheminFichier	Chaîne	➡	Nom ou chemin d'accès complet du fichier de base de données à ouvrir (.4db, .4dc, .4dbase ou .4dlink)

Description

La commande **OUVRIR BASE** referme la base de données 4D courante et ouvre à la volée la base désignée par le paramètre *cheminFichier*. Cette commande est utile dans le cadre de tests automatiques ou pour rouvrir une base après compilation.

Dans le paramètre *filePath*, passez le nom le chemin d'accès complet de la base de données à ouvrir. Vous pouvez utiliser un fichier ayant l'une des extensions suivantes :

- .4db (fichier de structure interprété),
- .4dc (fichier de structure compilé),
- .4dbase (package OS X),
- .4dlink (fichier de raccourci).

Si vous passez uniquement un nom de fichier, il doit être placé au même niveau que le fichier de structure de la base courante.

Si le chemin d'accès est valide, 4D quitte la base ouverte et ouvre la base spécifiée. En mode monoposte, la **Méthode base Sur fermeture** de la base refermée et la **Méthode base Sur ouverture** de la base ouverte sont successivement appelées.

Attention : Comme la commande entraîne la fermeture de l'application courant avant d'ouvrir la base spécifiée, il est déconseillé de l'appeler dans la **Méthode base Sur ouverture** ou dans une méthode appelée par cette méthode base.

La commande est exécutée de manière asynchrone : après son appel, 4D continue d'exécuter le reste de la méthode. Ensuite, l'application se comporte comme si la commande **Quitter** du menu **Fichier** avait été sélectionnée : les boîtes de dialogue d'ouverture sont annulées, tous les process ouverts sont tenus de se terminer en moins de dix secondes, etc.

Si le fichier de base cible n'est pas trouvé ou est invalide, une erreur système standard du gestionnaire de fichiers est retournée et 4D ne fait rien.

Cette commande peut être exécutée depuis une base de données standard uniquement. Si elle est appelée depuis une application fusionnée (monoposte ou serveur), l'erreur -10509 "Impossible d'ouvrir la base de données" est retournée.

Exemple

```
OUVRIR BASE("C:\\databases\\Invoices\\Invoices.4db")
```

OUVRIR CENTRE DE SECURITE

OUVRIR CENTRE DE SECURITE
Ne requiert pas de paramètre

Description

La commande **OUVRIR CENTRE DE SECURITE** provoque l'affichage de la fenêtre du Centre de sécurité et de maintenance (CSM).

En fonction des privilèges d'accès de l'utilisateur courant, certaines fonctions proposées dans la fenêtre pourront être désactivées.

Note : Cette commande fonctionne sur le même principe qu'un appel à **DIALOGUE** avec le paramètre *: le CSM est affiché dans une fenêtre et la commande rend immédiatement la main au code 4D. Si le process courant se termine, la fenêtre est automatiquement fermée en simulant un **NE PAS VALIDER**. Vous devez donc gérer son affichage via le code du process en cours d'exécution.

OUVRIR FENETRE ADMINISTRATION

OUVRIR FENETRE ADMINISTRATION
Ne requiert pas de paramètre

Description

La commande **OUVRIR FENETRE ADMINISTRATION** affiche la fenêtre d'administration du serveur sur le poste qui l'exécute. La fenêtre d'administration de 4D Server permet de visualiser les paramètres courants et d'effectuer diverses opérations de maintenance (cf. Guide de référence de 4D Server). A compter de la version 11 de 4D Server, cette fenêtre peut être affichée depuis un poste client :

Cette commande doit être appelée dans le contexte d'une application 4D connectée ou d'un 4D Server. Elle ne fait rien si :

- elle est appelée dans une application 4D en mode local,
- elle est exécutée par un utilisateur autre que le Super_Utilisateur ou l'Administrateur (dans ce cas, l'erreur -9991 est générée, cf. section **Erreurs de la base de données (-10602 -> 4004)**).

Exemple

Voici le code d'un bouton d'administration :

```
Si(Type application=4D mode local)
   OUVRIR CENTRE DE SECURITE
   ...
Fin de si
Si(Type application=4D mode distant)
   OUVRIR FENETRE ADMINISTRATION
   ...
Fin de si
Si(Type application=4D Server)
   OUVRIR CENTRE DE SECURITE
   ...
Fin de si
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1. Dans le cas contraire, elle prend la valeur 0.

OUVRIR FENETRE PROPRIETES

OUVRIR FENETRE PROPRIETES (sélecteur {; accès}{; typePropriétés})

Paramètre	Type		Description
sélecteur	Chaîne	➡	Clé désignant un thème ou une page de la boîte de dialogue des Préférences ou des Propriétés de la base
accès	Booléen	➡	Vrai=Verrouiller les autres pages de la boîte de dialogue, Faux ou omis=Laisser actives les autres pages de la boîte de dialogue
typePropriétés	Entier long	➡	0 ou omis = Propriétés structure (mode standard), 1 = Propriétés utilisateur

Description

La commande **OUVRIR FENETRE PROPRIETES** provoque l'ouverture de la boîte de dialogue des Préférences 4D ou des Propriétés de la base courante et l'affichage des paramètres ou de la page correspondant à la clé passée dans le paramètre *sélecteur*.

Le paramètre *sélecteur* doit contenir une "clé" désignant la boîte de dialogue et la page à ouvrir. Cette clé est construite de la manière suivante : */Dialogue{/Page{/Paramètres}}*. *Dialogue* indique la boîte de dialogue à afficher : vous pouvez passer "4D" (Préférences) ou "Database" (Propriétés de la base). Par exemple, pour désigner la page Compilateur des Propriétés de la base, *sélecteur* doit contenir "*/Database/Compiler*". La liste des clés est fournie ci-dessous. Si vous passez uniquement une barre oblique ("/") dans *sélecteur*, la commande affiche la première page de la boîte de dialogue des Propriétés de la base.

Le paramètre *accès* vous permet de contrôler les actions de l'utilisateur dans la boîte de dialogue des Préférences ou des Propriétés de la base en verrouillant les autres pages. Typiquement, vous pouvez souhaiter laisser l'utilisateur personnaliser certains paramètres, mais éviter que les autres puissent être modifiés. Dans ce cas, passez Vrai dans le paramètre *accès* : seule la page désignée par le paramètre *sélecteur* sera active et modifiable, l'accès à toutes les autres pages sera verrouillé (les clics sur les boutons de la barre de navigation seront sans effet). Si vous passez Faux ou omettez le paramètre *accès*, toutes les pages de la boîte de dialogue seront accessibles sans restriction.

Le paramètre *typePropriétés* est pris en compte dans les bases configurées en mode "Propriétés utilisateur" uniquement (dans ce mode, des "propriétés utilisateur" ou des "propriétés utilisateur pour le fichier de données" personnalisées sont générées dans un fichier externe et utilisées à la place des propriétés standard, cf. section **Utiliser des propriétés utilisateur** dans le manuel *Mode Développement*). Dans ce contexte, ce paramètre vous permet d'indiquer si vous souhaitez accéder à la boîte de dialogue des "propriétés structure", des "propriétés utilisateur" ou des '"propriétés utilisateur pour fichier de données". Vous pouvez passer une des constantes suivantes, placées dans le thème **Environnement 4D** :

Constante	Type	Valeur	Comment
Propriétés structure	Entier long	0	Accès aux "propriétés structure" (valeur par défaut si le paramètre est omis). Dans ce mode, les valeurs de *sélecteur* utilisables sont identiques à celles du mode standard.
Propriétés utilisateur	Entier long	1	Accès aux "propriétés utilisateur". Dans ce mode, seules certaines clés sont utilisables dans le paramètre *sélecteur*.
Propriétés utilisateur pour données	Entier long	2	Accès aux "propriétés utilisateur pour données", c'est-à-dire les propriétés utilisateur stockées au même niveau que le fichier de données. Dans ce mode, seules certaines clés peuvent être utilisées avec le paramètre *sélecteur* (même sous-ensemble que les propriétés utilisateur).

Si vous passez une clé invalide, la première page de la boîte de dialogue des Propriétés de la base est affichée.

Clés de chemins (mode standard)

Voici la liste des clés utilisables dans le paramètre *sélecteur* en mode standard, c'est-à-dire avec les "propriétés structure" :

/4D
/4D/General
/4D/Structure
/4D/Form editor
/4D/Method editor
/4D/Client-Server
/4D/Shortcuts
/Database
/Database/General
/Database/Mover
/Database/Interface
/Database/Interface/Developper
/Database/Interface/User
/Database/Interface/Shortcuts
/Database/Compiler
/Database/Database
/Database/Database/Data storage
/Database/Database/Memory and cpu
/Database/Database/International
/Database/Backup
/Database/Backup/Scheduler
/Database/Backup/Configuration
/Database/Backup/Backup and restore
/Database/Client-Server
/Database/Client-Server/Network
/Database/Client-Server/IP configuration
/Database/Web
/Database/Web/Config
/Database/Web/Options 1
/Database/Web/Options 2
/Database/Web/Log format
/Database/Web/Log scheduler
/Database/Web/Webservices
/Database/SQL
/Database/php

/Database/Compatibility
/Database/Security

Note de compatibilité : La commande continue de fonctionner avec les clés définies pour les versions 11.x et précédentes de 4D, la correspondance est établie automatiquement par le programme 4D. Il est toutefois conseillé de remplacer les anciens appels par les clés décrites ci-dessus.

Clés de chemins (mode Propriétés utilisateur)

Voici la liste des clés utilisables dans le paramètre *sélecteur* en mode "propriétés utilisateur" :

/Database
/Database/Interface
/Database/Database/Memory and cpu
/Database/Client-Server
/Database/Client-Server/Network
/Database/Client-Server/IP configuration
/Database/Web
/Database/Web/Config
/Database/Web/Options 1
/Database/Web/Options 2
/Database/Web/Log format
/Database/Web/Log scheduler
/Database/Web/Webservices
/Database/SQL
/Database/php

Exemple 1

Ouverture de la page “Méthodes” des Préférences 4D :

```
OUVRIR FENETRE PROPRIETES("/4D/Method editor")
```

Exemple 2

Accès aux paramétrages des raccourcis clavier dans les Propriétés de la base avec verrouillage des autres propriétés :

```
OUVRIR FENETRE PROPRIETES("/Database/Interface/Shortcuts";Vrai)
```

Exemple 3

Ouverture des Propriétés de la base sur la première page des Propriétés de la base :

```
OUVRIR FENETRE PROPRIETES("/")
```

Exemple 4

Accès à la page Interface des Propriétés de la base en mode "Propriétés utilisateur" :

```
OUVRIR FENETRE PROPRIETES("/Database/Interface";1)
```

Variables et ensembles système

Si la boîte de dialogue des préférences/propriétés est validée, la variable système OK retourne 1 ; si elle est annulée, OK retourne 0.

OUVRIR FICHIER DONNEES

OUVRIR FICHIER DONNEES (cheminAccès)

Paramètre	Type		Description
cheminAccès	Chaîne	➡	Nom ou chemin d'accès complet du fichier de données à ouvrir

Description

La commande **OUVRIR FICHIER DONNEES** permet de changer à la volée le fichier de données ouvert par l'application 4D.

Vous passez dans le paramètre *cheminAccès* le nom ou le chemin d'accès complet du fichier de données à ouvrir (fichier suffixé ".4DD"). Si vous passez uniquement un nom de fichier, il doit être placé à côté du fichier de structure de la base.

Si ce chemin d'accès désigne un fichier de données valide, 4D quitte la base en cours et la rouvre avec le fichier de données spécifié. En mode monoposte, la **Méthode base Sur fermeture** et la **Méthode base Sur ouverture** sont successivement appelées.

Attention : Comme cette commande provoque la fermeture préalable de l'application, elle doit être utilisée avec précaution dans la **Méthode base Sur ouverture** ou une méthode appelée par cette méthode base afin de ne pas générer de boucle sans fin.

La commande est exécutée de manière asynchrone : après son appel, 4D continue d'exécuter le reste de la méthode. Ensuite, l'application se comporte comme si la commande **Quitter** avait été sélectionnée dans le menu **Fichier** : les boîtes de dialogue ouvertes sont annulées, les process ouverts ont 10 secondes pour se terminer avant d'être tués, etc.

Avant de lancer l'opération, la commande teste la validité du fichier de données spécifié. En outre, si le fichier a déjà été ouvert, la commande vérifie qu'il correspond bien à la structure courante.

Si vous passez une chaîne vide dans *cheminAccès*, la commande rouvre la base sans changer de fichier de données.

4D Server : A compter de 4D v13, cette commande peut être exécutée avec 4D Server. Dans ce contexte, elle effectue en interne un appel à **QUITTER 4D** sur le serveur (entraînant l'apparition, sur chaque poste distant, d'une boîte de dialogue signalant que le serveur est en train de quitter) avant d'ouvrir le fichier désigné.

Exemple

Dans le contexte du déploiement d'une application fusionnée, vous souhaitez ouvrir ou créer le fichier de données utilisateur dans la méthode base Sur ouverture. Cet exemple utilise le fichier de données par défaut (cf. **Gestion du fichier de données dans les applications finales**) :

```
Si(Fichier donnees="@default.4dd")
   Si(Type version?? Application fusionnée)
      Si(Fichier donnees verrouille)
         $dataPath:=Dossier 4D(Dossier 4D actif)+"data.4dd"
 //Si un fichier de données local existe déjà
         Si(Tester chemin acces($dataPath)=Est un document)
            OUVRIR FICHIER DONNEES($dataPath) // on l'ouvre
         Sinon
            CREER FICHIER DONNEES($dataPath) //on le crée
         Fin de si
      Fin de si
   Fin de si
Fin de si
```

QUITTER 4D

QUITTER 4D {(délai)}

Paramètre	Type		Description
délai	Entier long	➡	Délai (secondes) avant que le serveur ne quitte

Description

La commande **QUITTER 4D** vous permet de quitter l'application 4D courante et de retourner sur le Bureau du système d'exploitation.

Le mécanismes mis en jeu par la commande sont différents suivant qu'elle est exécutée sur 4D (mode local ou distant) ou 4D Server (procédure stockée).

Avec 4D en mode local ou distant

Après un appel à **QUITTER 4D**, l'exécution du process courant est stoppée, puis 4D effectue les opérations suivantes :

- Si une **Méthode base Sur fermeture** existe, 4D l'exécute dans un nouveau process local. Par exemple, vous pouvez utiliser cette méthode base pour informer les autres process, via la communication interprocess, qu'ils doivent être fermés (s'ils sont en saisie de données) ou stopper l'exécution des opérations démarrées dans la **Méthode base Sur ouverture** (connexion de 4D à un autre serveur de bases de données). Notez que 4D quittera dans tous les cas : la **Méthode base Sur fermeture** peut assurer le nettoyage et la fermeture de toutes les opérations que vous voulez, mais la fermeture de la base est inéluctable.
- S'il n'existe pas de **Méthode base Sur fermeture**, 4D ferme tous les process un par un, sans distinction.

Si l'utilisateur est en saisie de données, les enregistrements seront annulés et non validés.
Si vous voulez permettre à l'utilisateur de sauvegarder ses modifications effectuées dans les fenêtres du process courant, vous pouvez utiliser la communication interprocess pour indiquer à tous les autres process utilisateur que la base est sur le point d'être quittée. Pour cela, vous pouvez adopter deux stratégies :

- Effectuer ces opérations depuis le process courant avant d'appeler **QUITTER 4D**.
- Traiter ces opérations depuis la **Méthode base Sur fermeture**.

Une troisième stratégie est également possible. Avant d'appeler **QUITTER 4D**, vous testez si une fenêtre nécessite une validation. Si c'est le cas, vous demandez à l'utilisateur de valider ou d'annuler cette fenêtre puis de choisir Quitter de nouveau. Cependant, du point de vue purement "interface utilisateur", les deux premières solutions sont préférables.

Note : Le paramètre *délai* n'est pas utilisable avec 4D en mode local ou distant.

Avec 4D Server (procédure stockée)

La commande **QUITTER 4D** peut être exécutée sur le poste serveur, dans une procédure stockée.
Dans ce cas, elle admet le paramètre optionnel *délai*. Ce paramètre permet d'allouer à 4D Server un délai d'attente avant que l'application ne quitte réellement, laissant ainsi aux postes clients le temps de se déconnecter. Vous devez passer dans *délai* une valeur en secondes.
Ce paramètre n'est pris en compte que dans le cadre d'une exécution sur le poste serveur. Avec 4D en mode local ou distant, il est ignoré.
Si vous ne passez pas le paramètre *délai*, 4D Server attendra que tous les postes clients soient déconnectés avant de quitter.
A la différence de 4D, le traitement de **QUITTER 4D** par 4D Server est asynchrone : la méthode dans laquelle la commande est appelée n'est pas interrompue après son exécution.

Si une **Méthode base Sur arrêt serveur** existe, elle est exécutée à l'issue du délai défini par le paramètre *délai*, ou de la déconnexion de tous les clients, suivant vos paramétrages.

L'action de la commande **QUITTER 4D** utilisée dans une procédure stockée est équivalente à celle de la commande **Quitter** du menu **Fichier** de 4D Server : elle provoque l'apparition, sur chaque poste client, d'une boîte de dialogue signalant que le serveur est sur le point de quitter.

Exemple

La méthode projet suivante est associée à la commande **Quitter** du menu **Fichier**.

```
   ` Méthode projet M_QUITTER

CONFIRMER("Etes-vous certain de vouloir quitter ?")
Si(OK=1)
   QUITTER 4D
Fin de si
```

REDEMARRER 4D

REDEMARRER 4D {(délai {; message})}

Paramètre	Type		Description
délai	Entier long	➡	Délai (secondes) avant que 4D ne redémarre
message	Chaîne	➡	Texte à afficher sur les postes clients

Description

La commande **REDEMARRER 4D** provoque le redémarrage de l'application 4D courante.

Cette commande est principalement destinée à une utilisation dans le contexte d'une application fusionnée (client/serveur ou monoposte) et en combinaison avec la commande **FIXER DOSSIER MISE A JOUR**. Dans ce cas, le processus de mise à jour automatique est enclenché : la nouvelle version de l'application désignée par **FIXER DOSSIER MISE A JOUR** remplace automatiquement la version courante au moment du redémarrage consécutif à **REDEMARRER 4D**. Le chemin d'accès au fichier de données est mémorisé et est automatiquement utilisé.

Si aucune information de mise à jour n'a été définie via la commande **FIXER DOSSIER MISE A JOUR** dans la session courante, la commande redémarre simplement l'application 4D avec les fichiers de structure et de données courants.

Le paramètre *délai* permet de différer le redémarrage de l'application afin de laisser aux postes clients le temps de se déconnecter. Vous devez passer une valeur en secondes dans *délai*. Si vous omettez ce paramètre, l'application serveur attendra que toutes les applications clientes se soient déconnectées, dans un délai maximum de 10 minutes. Au-delà, toutes les applications clientes sont automatiquement déconnectées.

Note : Les paramètres *délai* et *message* sont uniquement pris en compte avec les applications serveur (ils sont ignorés lorsque la commande est exécutée sur une application monoposte ou sur un 4D distant).

Le paramètre optionnel *message* vous permet d'afficher un message personnalisé aux applications clientes connectées.

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et l'application redémarre. Vous pouvez intercepter les erreurs éventuellement générées par la commande à l'aide d'une méthode installée via la commande **APPELER SUR ERREUR**.

Type application

Type application -> Résultat

Paramètre	Type		Description
Résultat	Entier long	➲	Valeur numérique représentant le type de l'application

Description

La fonction **Type application** renvoie une valeur numérique qui représente le type de l'environnement 4D que vous utilisez. 4D vous fournit les constantes prédéfinies suivantes, placées dans le thème **Environnement 4D** :

Constante	Type	Valeur
4D Desktop	Entier long	3
4D mode distant	Entier long	4
4D mode local	Entier long	0
4D Server	Entier long	5
4D Volume Desktop	Entier long	1

Note : *4D Desktop* intègre certaines offres de déploiement, comme par exemple "4D SQL Desktop".

Exemple

Quelque part dans votre code, ailleurs que dans la **Méthode base Sur démarrage serveur**, vous voulez vérifier si l'utilisateur a ouvert la base avec 4D Server. Pour cela, vous pouvez écrire les lignes de code suivantes :

```
Si(Type application=4D Server)
  ` Exécuter des actions nécessaires
Fin de si
```

Type version

Type version -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Version de démonstration ou standard, Version 64 bits ou 32 bits, Base 4D ou Application fusionnée

Description

La commande **Type version** retourne une valeur numérique qui représente le type de version de 4D ou de 4D Server que vous utilisez. 4D vous fournit les constantes prédéfinies suivantes, placées dans le thème **Environnement 4D** :

Constante	Type	Valeur	Comment
Application fusionnée	Entier long	2	La version est une application fusionnée avec 4D Volume Desktop
Version 64 bits	Entier long	1	
Version de démonstration	Entier long	0	

Note : Dans les versions actuelles de 4D, le mode démonstration n'est pas disponible.

Type version retourne une valeur sous forme de *champ de bits*, il est nécessaire d'utiliser les opérateurs sur les bits pour l'interpréter (cf. exemples).

NOTE DE COMPATIBILITE : Dans les versions de 4D antérieures à la 13.2, un jeu de constantes différent était proposé pour cette commande ; ces constantes ne permettaient pas de traiter correctement tous les cas, elles ont donc été modifiées. Ce changement nécessite l'adaptation de votre code (cf. exemple). Toutefois, si vous souhaitez conserver le fonctionnement précédent, il vous suffit de remplacer les constantes dans votre code existant par leur ancienne valeur : 2 pour Version 64 bits, 1 pour Version de démonstration, 0 pour Version standard.

Exemple 1

Votre application 4D contient du code spécifique en fonction de la version. Vous pouvez connaître l'environnement d'exécution avec le code suivant :

```
Si(Type version?? Version 64 bits)
 //Nous sommes dans une version 64 bits
Sinon
 // Nous sommes dans une version 32 bits
Fin de si
```

Exemple 2

Ce test permet d'exécuter du code différent selon que la version est une application fusionnée ou une base ouverte par 4D / 4D Server :

```
Si(Type version?? Application fusionnée)
       // Nous sommes dans une application fusionnée
Sinon
       // Nous sommes dans une base de données exécutée par 4D
Fin de si
```

VERIFIER FICHIER DONNEES

VERIFIER FICHIER DONNEES (cheminStructure ; cheminDonnées ; objets ; options ; méthode {; tabTables {; tabChamps}})

Paramètre	Type		Description
cheminStructure	Texte	⇒	Chemin d'accès du fichier de structure de la base à vérifier
cheminDonnées	Texte	⇒	Chemin d'accès du fichier de données de la base à vérifier
objets	Entier long	⇒	Objets à vérifier
options	Entier long	⇒	Options de vérification
méthode	Texte	⇒	Nom de la méthode 4D de rétroappel
tabTables	Tableau entier long	⇒	Numéros des tables à vérifier
tabChamps	Tableau entier 2D, Tableau entier long 2D, Tableau réel 2D	⇒	Numéros des index à vérifier

Description

La commande **VERIFIER FICHIER DONNEES** effectue une vérification structurelle des objets contenus dans le fichier de données 4D désigné par *cheminStructure* et *cheminDonnées*.

Note : Pour plus d'informations sur le processus de vérification des données, reportez-vous au manuel Mode Développement.

cheminStructure désigne le fichier de structure (compilé ou non) associé au fichier de données à vérifier. Il peut s'agir du fichier de structure ouvert ou de tout autre fichier de structure. Vous devez passer un chemin d'accès complet, exprimé avec la syntaxe du système d'exploitation. Vous pouvez également passer une chaîne vide, dans ce cas une boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de désigner le fichier de structure à utiliser.

cheminDonnées désigne un fichier de données 4D (.4DD). Il doit correspondre au fichier de structure défini par le paramètre *cheminStructure*. Attention, vous pouvez désigner le fichier de structure courant mais le fichier de données ne doit pas être le fichier courant (ouvert). Pour vérifier le fichier de données ouvert, utilisez la commande **VERIFIER FICHIER DONNEES OUVERT**. Si vous tentez de vérifier le fichier de données courant avec la commande **VERIFIER FICHIER DONNEES**, une erreur est générée.
Le fichier de données désigné est ouvert en lecture seulement. Vous devez veiller à ce qu'aucune application n'accède à ce fichier en écriture, sinon les résultats de la vérification pourront être faussés.
Vous pouvez passer dans le paramètre *cheminDonnées* une chaîne vide, un nom de fichier ou un chemin d'accès complet, exprimé dans la syntaxe du système d'exploitation. Si vous passez une chaîne vide, la boîte de dialogue standard d'ouverture de fichier apparaît, permettant à l'utilisateur de désigner le fichier à vérifier (à noter dans ce cas qu'il n'est pas possible de sélectionner le fichier de données courant). Si vous passez uniquement un nom de fichier de données, 4D le recherchera au même niveau que le fichier de structure défini.

Le paramètre *objets* permet de désigner le(s) type(s) d'objets à vérifier. Deux types d'objets peuvent être vérifiés : les enregistrements et les index. Utilisez les constantes suivantes, placés dans le thème **Maintenance fichier de données** :

Constante	Type	Valeur	Comment
Tout vérifier	Entier long	16	
Vérifier enregistrements	Entier long	4	
Vérifier index	Entier long	8	Cette option contrôle la cohérence physique des index, sans lien avec les données. Elle signale des clés invalides mais ne permet pas de détecter les clés dupliquées (deux index pointant vers le même enregistrement). Ce type d'erreur ne peut être détecté qu'avec l'option Tout vérifier.

Pour vérifier les enregistrements et les index, passez le cumul *Vérifier enregistrements+Vérifier index*. La valeur 0 (zéro) permet également d'obtenir le même résultat. L'option Tout vérifier effectue la vérification interne la plus complète. Cette vérification est compatible avec la création d'un historique.

Le paramètre *options* permet de définir les options de vérification. Les options suivantes sont disponibles, accessibles via des constantes du thème **Maintenance fichier de données** :

Constante	Type	Valeur	Comment
Ne pas créer d'historique	Entier long	16384	En principe, la commande crée un fichier d'historique au format xml (reportez-vous à la fin de la description de la commande). Vous pouvez annuler ce fonctionnement en passant cette option.
Nom historique avec date heure	Entier long	262144	Lorsque cette option est passée, le nom du fichier d'historique généré contiendra la date et l'heure de sa création, par conséquent il ne remplacera aucun fichier d'historique éventuellement déjà généré. Par défaut, si cette option n'est pas passée, le nom du fichier n'est pas horodaté, et chaque nouveau fichier généré remplace le précédent.

Pour créer l'historique, passez 0 dans *options*.

Le paramètre *méthode* permet de définir une méthode de rétro-appel qui sera régulièrement appelée durant la vérification. Si vous passez une chaîne vide, aucune méthode n'est appelée. Si la méthode passée n'existe pas, la vérification n'est pas effectuée, une erreur est générée et la variable OK prend la valeur 0. Lorsqu'elle est appelée, cette méthode reçoit jusqu'à 5 paramètres en fonction des objets vérifiés et du type d'événement à l'origine de l'appel (cf. tableau des appels). Vous devez impérativement déclarer ces paramètres dans la méthode :

$1	Entier long	Type de message (cf. tableau)
$2	Entier long	Type d'objet
$3	Texte	Message
$4	Entier long	Numéro de table
$5	Entier long	Réservé

Le tableau suivant décrit le contenu des paramètres en fonction du type d'événement :

Evénement	$1 (Entier long)	$2 (Entier long)	$3 (Texte)	$4 (Entier long)	$5 (Entier long)
Message	1	0	Message de progression	Pourcentage réalisé (0-100)	Réservé
Vérification terminée(*)	2	Type d'objet(**)	Texte du message OK	Numéro de table ou d'index	Réservé
Erreur	3	Type d'objet(**)	Texte du message d'erreur	Numéro de table ou d'index	Réservé
Fin d'exécution	4	0	DONE	0	Réservé
Warning	5	Type d'objet(**)	Texte du message d'erreur	Numéro de table ou d'index	Réservé

(*) L'événement *Vérification terminée* ($1=2) n'est jamais renvoyé lorsque le mode de vérification est Tout vérifier. Il n'est utilisé qu'en mode Vérifier enregistrements ou Vérifier index.

(**) *Type d'objet* : Lorsqu'un objet est vérifié, un message "terminé" ($1=2), erreur ($1=3) ou warning ($1=5) peut être envoyé. Le type d'objet retourné dans $2 peut être l'un des suivants :

- 0 = indéterminé
- 4 = enregistrement
- 8 = index
- 16 = objet structure (contrôle préliminaire du fichier de données).

Cas particulier : lorsque $4 = 0 pour $1 = 2, 3 ou 5, le message ne concerne pas une table mais le fichier de données dans son ensemble.

La méthode de rétro-appel doit également retourner une valeur dans $0 (Entier long), permettant de contrôler l'exécution de l'opération :

- si $0 = 0, l'opération continue normalement
- si $0 = -128, l'opération est stoppée sans erreur générée
- si $0 = autre valeur, l'opération est stoppée et la valeur passée dans $0 est retournée en tant que numéro d'erreur. Cette erreur peut être interceptée par une méthode d'appel sur erreur.

Note : Il n'est pas possible d'interrompre l'exécution via $0 après que l'événement *Fin d'exécution* ($4=1) a été généré.

Deux tableaux facultatifs peuvent également être utilisés par la commande :

- Le tableau *tabTables* contient les numéros des tables dont les enregistrements doivent être vérifiés. Il permet de limiter la vérification à certaines tables. Si ce paramètre n'est pas passé ou si le tableau est vide et que le paramètre *objets* contient Vérifier enregistrements, toutes les tables sont vérifiées.
- Le tableau *tabChamps* contient les numéros des champs indexés dont les index doivent être vérifiés. Si ce paramètre n'est pas passé ou si le tableau est vide et que le paramètre *objets* contient Vérifier index, tous les index sont vérifiés. La commande ignore les champs non indexés. Si un champ contient plusieurs index, tous les index sont vérifiés. Si un champ fait partie d'un index composite, la totalité de l'index est vérifiée.
  Vous devez passer un tableau 2D dans *tabChamps*. Pour chaque ligne du tableau :
  - l'élément {0} contient le numéro de la table,
  - les autres éléments {1...n} contiennent les numéros des champs.

Par défaut, la commande **VERIFIER FICHIER DONNEES** crée un fichier d'historique au format xml (si vous n'avez pas passé l'option Ne pas créer d'historique, cf. paramètre *options*). Son nom est basé sur celui du fichier de structure de la base courante et il est également placé dans le dossier **Logs** de cette base. Par exemple, pour un fichier de structure nommé "myDB.4db", le fichier d'historique sera nommé "myDB_Verify_Log.xml".
Si vous avez passé l'option Nom historique avec date heure, le nom du fichier d'historique inclut la date et l'heure de sa création sous la forme "AAAA-MM-JJ HH-MM-SS", ce qui donne par exemple : "myDB_Verify_Log_2015-09-27 15-20-35.xml". Ce principe permet d'éviter que chaque nouveau fichier d'historique écrase le précédent, mais pourra nécessiter ultérieurement une action manuelle afin de supprimer les fichiers superflus.
Quelle que soit l'option sélectionnée, dès lors qu'un fichier d'historique est généré, son chemin est retourné dans la variable système *Document* à l'issue de l'exécution de la commande.

Exemple 1

Vérification simple des données et des index :

```
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Vérifier index+Vérifier enregistrements;Ne pas créer dhistorique;"")
```

Exemple 2

Vérification complète avec historique :

```
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Tout vérifier;0;"")
```

Exemple 3

Vérification des enregistrements uniquement :

```
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Vérifier enregistrements;0;"")
```

Exemple 4

Vérification des enregistrements des tables 3 et 7 uniquement :

```
TABLEAU ENTIER LONG($tnumTables;2)
$tnumTables{1}:=3
$tnumTables{2}:=7
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Vérifier enregistrements;0;"FollowScan";$tnumTables)
```

Exemple 5

Vérification d'index spécifiques (index du champ 1 de la table 4 et index des champs 2 et 3 de la table 5) :

```
TABLEAU ENTIER LONG($tnumTables;0) `non utilisé mais obligatoire
TABLEAU ENTIER LONG($tindex;2;0) `2 lignes (colonnes ajoutées ensuite)
$tindex{1}{0}:=4 ` numéro de table dans l'élément 0
AJOUTER A TABLEAU($tindex{1};1) ` numéro du 1er champ à vérifier
$tindex{2}{0}:=5 ` numéro de table dans l'élément 0
AJOUTER A TABLEAU($tindex{2};2) ` numéro du 1er champ à vérifier
AJOUTER A TABLEAU($tindex{2};3) ` numéro du 2e champ à vérifier
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Vérifier index;0;"FollowScan";$tnumTables;$tindex)
```

Variables et ensembles système

Si la méthode de rétro-appel passée n'existe pas, la vérification n'est pas effectuée, une erreur est générée et la variable OK prend la valeur 0. Si un fichier d'historique a été généré, son chemin complet est retourné dans la variable système Document.

VERIFIER FICHIER DONNEES OUVERT

VERIFIER FICHIER DONNEES OUVERT {(objets ; options ; méthode {; tabTables {; tabChamps}})}

Paramètre	Type		Description
objets	Entier long	⇒	Objets à vérifier
options	Entier long	⇒	Options de vérification
méthode	Texte	⇒	Nom de la méthode 4D de rétro-appel
tabTables	Tableau entier long	⇒	Numéros des tables à vérifier
tabChamps	Tableau entier 2D, Tableau entier long 2D, Tableau réel 2D	⇒	Numéros des index à vérifier

Description

La commande **VERIFIER FICHIER DONNEES OUVERT** effectue une vérification structurelle des objets contenus dans le fichier de données actuellement ouvert par 4D.

Cette commande a un fonctionnement identique à celui de la commande **VERIFIER FICHIER DONNEES**, à la différence près qu'elle s'applique uniquement au fichier de données courant de la base de données ouverte. Elle ne nécessite donc pas de paramètres désignant la structure et les données. Reportez-vous à la commande **VERIFIER FICHIER DONNEES** pour la description des paramètres.

Si vous passez directement la commande **VERIFIER FICHIER DONNEES OUVERT** sans aucun paramètre, la vérification est effectuée avec les valeurs par défaut des paramètres :

- *objets* = Tout vérifier (= valeur 16)
- *options* = 0 (l'historique est créé mais n'est pas horodaté)
- *méthode* = ""
- *tabTables* et *tabChamps* sont omis.

Lorsque cette commande est exécutée, le cache de données est écrit sur le disque et toutes les opérations accédant aux données sont bloquées durant la vérification.

Si un fichier d'historique a été généré, son chemin complet est retourné dans la variable système *Document*.

Variables et ensembles système

Si la méthode de rétro-appel passée n'existe pas, la vérification n'est pas effectuée, une erreur est générée et la variable OK prend la valeur 0. Si un fichier d'historique a été généré, son chemin complet est retourné dans la variable système Document.

Version application

Version application {(numBuild {; *})} -> Résultat

Paramètre	Type		Description
numBuild	Entier long	⇐	Numéro de build
*	Opérateur	⇒	Si passé = numéro de version long Si omis = numéro de version court
Résultat	Chaîne	⇒	Numéro de version dans une chaîne encodée

Description

Version application retourne une chaîne encodée qui exprime le numéro de version de l'environnement 4D que vous utilisez.

Si vous ne passez pas le paramètre optionnel *, une chaîne de 4 caractères est retournée, formatée de la manière suivante :

Caractères	Description
1-2	Numéro de version
3	Numéro "R"
4	Numéro de révision

Si vous passez le paramètre optionnel *, une chaîne de 8 caractères est retournée, formatée de la manière suivante :

Caractères	Description
1	"F" représente une version finale
	"B" représente une version beta
	Les autres caractères représentent une version interne à 4D
2-3-4	Numéro de compilation interne à 4D
5-6	Numéro de version
7	Numéro "R"
8	Numéro de révision

Note de compatibilité (4D v14)

Attention, la numérotation est modifiée à compter des versions 14 de 4D :

- le **numéro "R"** est le numéro de version "R" de 4D, par exemple 3 pour la version R3 (contient 0 pour une version "bug fix"),
- le **numéro de révision** est le numéro de version "bug fix" de 4D (contient 0 pour une version "R").

Dans les versions précédentes de 4D, le numéro de version "R" était le numéro de mise à jour, il désignait la révision. Le numéro de révision était toujours 0.

Exemples pour un numéro de version court :

Versions	Valeur retournée	
4D v13.1	"1310"	*Précédent système de numérotation*
4D v14 R2	"1420"	Release R2
4D v14 R3	"1430"	Release R3
4D v14.1	"1401"	Première version "bug fix" de 4D v14
4D v14.2	"1402"	Seconde version "bug fix" de 4D v14

Exemples pour un numéro de version long :

Versions	Valeur retournée
4D v12.5 beta	"B0011250"
4D v14 R2 beta	"B0011420"
4D v14 R3 finale	"F0011430"
4D v14.1 beta	"B0011401"

La commande **Version application** peut retourner une information supplémentaire dans le paramètre optionnel *numBuild* : le numéro de "build" de la version courante de l'application 4D. Il s'agit d'un numéro de compilation interne qui peut être utile pour du versionning ou lors d'échanges avec les services techniques de 4D.

Note : Dans le cas des applications compilées et fusionnées avec 4D Volume Licence, le numéro de build retourné n'est pas significatif. Dans ce contexte, les informations de version sont gérées par le développeur.

Exemple 1

Cet exemple affiche le numéro de version de l'environnement 4D :

```
$vs4Dversion:=Version application
ALERTE("Vous utilisez la version "+Chaine(Num(Sous
chaine($vs4Dversion;1;2)))+"."+$vs4Dversion≤3≥+"."+$vs4Dversion≤4≥)
```

Exemple 2

Cet exemple teste si vous utilisez une version finale :

```
Si(Sous chaine(Version application(*);1;1)#"F")
   ALERTE("Veuillez vous assurer que vous utilisez une version finale de 4D avec cette base !")
   QUITTER 4D
Fin de si
```

Exemple 3

Le code suivant reconstitue le numéro de version de l'application et permet de distinguer les versions v14 "bug fix" des versions v14 "R" :

```
C_ENTIER LONG($Lon_build)
C_TEXTE($Txt_info;$Txt_major;$Txt_minor;$Txt_release;$Txt_version)

$Txt_version:=Version application($Lon_build)

$Txt_major:=$Txt_version[[1]]+$Txt_version[[2]] //numéro de version, p.e. 14
$Txt_release:=$Txt_version[[3]] //Rx
$Txt_minor:=$Txt_version[[4]] //.x

$Txt_info:="4D v"+$Txt_major
Si($Txt_release="0") //4D v14.x
   $Txt_info:=$Txt_info+Choisir($Txt_minor#"0";"."+$Txt_minor;"")

Sinon //4D v14 Rx
   $Txt_info:=$Txt_info+" R"+$Txt_release
Fin de si
```

_o_AJOUTER SEGMENT DE DONNÉES

_o_AJOUTER SEGMENT DE DONNÉES
Ne requiert pas de paramètre

Description

Note de compatibilité : Depuis la version 11 de 4D, les segments de données ne sont plus pris en charge (la taille du fichier de données est désormais illimitée). Lorsqu'elle est appelée, cette commande ne fait rien.

_o_LISTE SEGMENTS DE DONNÉES

_o_LISTE SEGMENTS DE DONNÉES (segments)

Paramètre	Type		Description
segments	Tableau chaîne	⇦	Noms longs des segments de données de la base

Note de compatibilité

Depuis la version 11 de 4D, les segments de données ne sont plus pris en charge (la taille du fichier de données est désormais illimitée). Cette commande retourne désormais systématiquement un tableau à un élément, contenant le chemin d'accès du fichier de données de la base.

Environnement système

- Authentification client courant
- COORDONNEES ECRAN
- Dossier systeme
- Dossier temporaire
- Ecran principal
- ENREGISTRER EVENEMENT
- FIXER POLICES RECENTES
- FIXER PROFONDEUR ECRAN
- Gestalt
- Hauteur barre de menus
- Hauteur ecran
- Largeur ecran
- LIRE FORMATAGE SYSTEME
- LISTE DES POLICES
- LISTE STYLES POLICE
- Nom de la machine
- Nom du possesseur
- Nombre ecrans
- OUVRIR SELECTEUR COULEUR
- OUVRIR SELECTEUR POLICE
- PROFONDEUR ECRAN
- PROPRIETES PLATE FORME
- Selectionner couleur RVB
- *_o_Nom de police*
- *_o_Numéro de police*

Authentification client courant

Authentification client courant {(domaine ; protocole)} -> Résultat

Paramètre	Type		Description
domaine	Texte	⇐	Nom du domaine
protocole	Texte	⇐	"Kerberos", "NTLM" ou chaîne vide
Résultat	Texte	⇒	Nom d'utilisateur de session retourné par Windows

Description

La commande **Authentification client courant** envoie au serveur Active Directory de Windows une requête d'authentification du client courant et, en cas de succès, retourne le nom d'utilisateur Windows de ce client (identifiant de session). Si l'authentification échoue, une chaîne vide est retournée.

Cette commande peut être utilisée uniquement dans le contexte d'une implémentation SSO sous Windows avec 4D Server. Pour plus d'informations, veuillez vous reporter à la section **Authentification unique (SSO) sous Windows**.

Généralement, le client et le serveur doivent être gérés par le même serveur Active Directory. Cependant, des configurations spécifiques peuvent être prises en charge, comme décrit dans le paragraphe **Configuration requise pour le SSO**.

La chaîne retournée par la commande doit être passée à votre module d'identification 4D. Vous pouvez ainsi déterminer automatiquement les droits d'accès du client en fonction de son identifiant de session Windows. Si vous définissez un "Utilisateur par défaut", vous pouvez implémenter une interface dans laquelle l'utilisateur n'a pas de ressaisir son identifiant -- la boîte de dialogue d'identification de l'utilisateur de 4D Server n'apparaît pas (voir exemple).

Optionnellement, la commande peut retourner deux paramètres de type texte :

- *domaine* : nom du domaine auquel appartient le client.
- *protocole* : nom du protocole utilisé par Windows pour authentifier l'utilisateur. Il peut contenir "Kerberos" ou "NTLM", en fonction des ressources disponibles. Si l'authentification a échoué, une chaîne vide ("") est retournée.

Ces paramètres peuvent être utilisés pour accepter ou rejeter les connexions si vous souhaitez filtrer les accès en fonction du domaine du client ou du protocole utilisé.

Niveau de sécurité de l'authentification

Le niveau de sécurité de l'authentification (c'est-à-dire le degré de confiance que vous pouvez avoir dans le nom d'utilisateur récupéré par la commande) dépend de la manière dont l'utilisateur a été identifié. Les valeurs retournées dans les différents paramètres de la commande **Authentification client courant** vous permettent de savoir quelles informations ont été utilisées pour l'identification et donc, le niveau de sécurité :

Nom d'utilisateur	domaine	protocole	Commentaire
Vide	Vide	Vide	La commande n'a pas pu récupérer d'information d'authentification relatives à l'utilisateur connecté.
Valeur reçue	Vide	NTLM	La valeur reçue est le nom d'utilisateur local, défini sur la machine locale.
Valeur reçue	Valeur reçue	NTLM	Le nom d'utilisateur retourné a été authentifié via le protocole NTLM dans le domaine retourné par le paramètre *domaine*. Dans ce cas, vous devez contrôler le domaine afin d'augmenter le niveau de sécurité. Comme certaines architectures comportent une forêt de domaines, vous devez en particulier vérifier que le domaine dans lequel l'utilisateur a été identifié est bien le domaine souhaité.
Valeur reçue	Valeur reçue	Kerberos	Le nom d'utilisateur retourné a été authentifié avec le protocole Kerberos dans le domaine souhaité. Cette configuration constitue le niveau de sécurité le plus élevé.

Pour plus d'informations sur les configurations, veuillez vous reporter au paragraphe **Configuration requise pour le SSO**.

Exemple

Dans votre base 4D Server, vous avez conçu un système de contrôle d'accès basé sur la fonctionnalité des utilisateurs et des groupes de 4D. Vous souhaitez configurer votre application de manière à ce que les utilisateurs 4D distants sous Windows puissent se connecter directement à 4D Server (sans qu'aucune boîte de dialogue de mot de passe ne s'affiche), tout en étant connectés avec leurs propres droits d'accès.

1. Dans la page "Sécurité" de la boîte de dialogue des Propriétés de la base, désignez un "Utilisateur par défaut" :

   Avec ce paramétrage, aucune boîte de dialogue d'identification n'est affichée pour les utilisateurs 4D distants qui se connectent au serveur -- tous les clients sont connectés par défaut en tant que "Bob".
2. Dans la **MissingRef**, ajoutez le code suivant afin d'authentifier l'utilisateur auprès de l'Active Directory:

```
 //Méthode base Sur ouverture connexion serveur
C_ENTIER LONG($0;$1;$2;$3)
$login:=Authentification client courant($domain;$protocol)
Si($login #"") //un nom d'utilisateur a bien été retourné
 //appelez votre méthode d'identification personnalisée
   $0:=CheckCredentials($login)
 //elle doit retourner 0 en cas de succès, -1 en cas d'erreur
Sinon
   $0:=-1 //rejeter la connexion
Fin de si
```

Note : Cet exemple constitue un scénario de base, qui doit être adapté à vos solutions. La méthode d'identification personnalisée de l'utilisateur 4D (***CheckCredentials*** dans l'exemple ci-dessus) peut être basée sur l'une des implémentations suivantes :

- réplication de l'Active Directory dans les noms d'utilisateurs et groupes de 4D, permettant une correspondance automatique,
- utilisation d'une table [Utilisateurs] personnalisée,
- utilisation des fonctions LDAP afin de récupérer les droits d'accès de l'utilisateur.

COORDONNEES ECRAN

COORDONNEES ECRAN (gauche ; haut ; droite ; bas {; écran})

Paramètre	Type		Description
gauche	Entier long	⇐	Coordonnée gauche de la zone de l'écran
haut	Entier long	⇐	Coordonnée supérieure de la zone de l'écran
droite	Entier long	⇐	Coordonnée droite de la zone de l'écran
bas	Entier long	⇐	Coordonnée inférieure de la zone de l'écran
écran	Entier long	⇒	Numéro de l'écran ou écran principal si omis

Description

La commande **COORDONNEES ECRAN** retourne dans les paramètres *gauche*, *haut*, *droite* et *bas* les coordonnées de l'écran spécifié dans le paramètre *écran*.

Si vous omettez le paramètre *écran*, **COORDONNEES ECRAN** retourne les coordonnées de l'écran principal.

Dossier systeme

Dossier systeme {(type)} -> Résultat

Paramètre	Type		Description
type	Entier long	⇨	Type de dossier système
Résultat	Chaîne	⇦	Chemin d'accès d'un dossier du système actif

Description

La fonction **Dossier systeme** retourne le chemin d'accès du dossier Système Windows ou Mac OS actif, ou le chemin d'accès d'un dossier particulier du système d'exploitation.

Le paramètre optionnel *type* vous permet d'indiquer le type de dossier dont vous souhaitez obtenir le chemin d'accès. Si vous ne passez pas ce paramètre, **Dossier systeme** retourne le chemin d'accès du dossier Système Windows ou Mac OS actif.

Vous passez dans *type* un code représentant le type de dossier. 4D fournit les constantes prédéfinies suivantes (placées dans le thème "**Dossier Système**") :

Constante	Type	Valeur	Comment
Applications ou Program Files	Entier long	16	
Bureau	Entier long	15	
Démarrage Win	Entier long	5	
Démarrage Win_Tous	Entier long	4	
Dossier documents	Entier long	17	Dossier "Documents" de l'utilisateur
Favoris Win	Entier long	14	
Menu Démarrer Win	Entier long	9	
Menu Démarrer Win_Tous	Entier long	8	
Polices	Entier long	1	
Préférences utilisateur	Entier long	3	
Préférences utilisateur_Tous	Entier long	2	
System Win	Entier long	12	
System32 Win	Entier long	13	
Système	Entier long	0	

Notes :

- Les constantes suffixées **Win** sont réservées à une utilisation sous Windows. Lorsqu'elles sont utilisées sous Mac OS, **Dossier systeme** retourne une chaîne vide.
- L'emplacement de certains dossiers peut être différent suivant le type de session ouverte par l'utilisateur. Les constantes 2 à 9 permettent de choisir si vous souhaitez obtenir le chemin d'accès du dossier spécifique à l'utilisateur courant (constantes simples) ou commun à tous les utilisateurs (constantes suivies de "Tous").

Dossier temporaire

Dossier temporaire -> Résultat

Paramètre	Type		Description
Résultat	Chaîne		Chemin d'accès au dossier temporaire

Description

La fonction **Dossier temporaire** retourne le chemin d'accès au dossier temporaire courant tel que défini par votre système d'exploitation.

Exemple

Reportez-vous à l'exemple de la commande **AJOUTER DONNEES AU CONTENEUR**.

Ecran principal

Ecran principal -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Numéro de l'écran contenant la barre de menus

Description

La commande **Ecran principal** retourne le numéro de l'écran dans lequel se trouve la barre de menus.
Note pour les utilisateurs Windows : Sous Windows, **Ecran principal** renvoie toujours 1.

ENREGISTRER EVENEMENT

ENREGISTRER EVENEMENT ({typeSortie ;} message {; importance})

Paramètre	Type		Description
typeSortie	Entier long	⇒	Type de sortie du message
message	Chaîne	⇒	Contenu du message
importance	Entier long	⇒	Niveau d'importance du message (Windows uniquement)

Description

La commande **ENREGISTRER EVENEMENT** vous permet de mettre en place un système personnalisé d'enregistrement des événements internes qui se produisent au cours de l'utilisation de votre application. Vous pouvez ainsi contrôler le déroulement d'une session de travail.

Passez dans le paramètre *message* les informations personnalisées à noter en fonction de l'événement.

Le paramètre facultatif *typeSortie* vous permet de préciser le canal de sortie emprunté par le *message*. Vous pouvez passer dans ce paramètre une des constantes suivantes, placées dans le thème **Journal d'événements** :

Constante	Type	Valeur	Comment
Vers historique commandes 4D	Entier long	3	Indique à 4D d'inscrire le *message* dans le fichier d'historique des commandes de 4D, si ce fichier a été activé. Ce fichier d'historique peut être activé à l'aide de la commande **FIXER PARAMETRE BASE** (sélecteur 34). **Note :** Les fichiers d'historique de 4D sont regroupés dans le dossier **Logs**, créé à côté du fichier de structure de la base (cf. commande **Dossier 4D**).
Vers historique diagnostic	Entier long	5	Indique à 4D d'inscrire le *message* dans le fichier de diagnostic de 4D, si ce fichier a été activé. Le fichier de diagnostic peut être activé à l'aide de la commande **FIXER PARAMETRE BASE** (sélecteur 79).
Vers historique requêtes 4D	Entier long	2	Indique à 4D d'inscrire le *message* dans le fichier d'historique des requêtes de 4D, si ce fichier a été activé
Vers message débogage	Entier long	1	Indique à 4D d'envoyer le *message* dans l'environnement de débogage du système. Le résultat dépend de la plate-forme : • sous Mac OS : la commande envoie le message à la Console • sous Windows : la commande envoie le message en tant que message de débogage. Pour pouvoir lire ce message, vous devez disposer de Microsoft Visual Studio ou de l'utilitaire DebugView pour Windows (http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx)
Vers observateur Windows	Entier long	0	Indique à 4D d'envoyer le *message* vers l'"Observateur d'événements" de Windows. Ce journal reçoit et stocke des messages en provenance des applications en cours d'exécution. Dans ce cas, vous pouvez définir le niveau d'importance du *message* via le paramètre *importance* (cf. ci-dessous). **Notes :** • Pour que cette fonctionnalité soit disponible, le service Observateur d'événements de Windows doit être démarré. • Sous Mac OS, la commande ne fait rien avec ce type de sortie.

Si vous ne passez pas le paramètre *typeSortie*, la valeur 0 (Vers observateur Windows) est utilisée par défaut.

Si vous avez défini un *typeSortie* de type Vers observateur Windows , vous pouvez attribuer au *message* un niveau d'importance via le paramètre facultatif *importance* afin de faciliter la lecture du journal d'événements. Il existe trois niveaux d'importance : Information, Avertissement et Erreur. 4D vous propose les constantes prédéfinies suivantes, placées dans le thème **Journal d'événements** :

Constante	Type	Valeur
Message d'erreur	Entier long	2
Message d'avertissement	Entier long	1
Message d'information	Entier long	0

Si vous ne passez pas le paramètre *importance* ou passez une valeur invalide, la valeur par défaut (0) est utilisée.

Exemple

Afin de conserver une trace des lancements de votre base sous Windows, vous pouvez écrire, dans la **Méthode base Sur ouverture** :

```
ENREGISTRER EVENEMENT(Vers observateur Windows;"Démarrage de la base Facturation")
```

A chaque ouverture de la base, cette information sera inscrite dans l'Observateur d'événements de Windows, avec le niveau d'importance 0.

FIXER POLICES RECENTES

FIXER POLICES RECENTES (tabPolices)

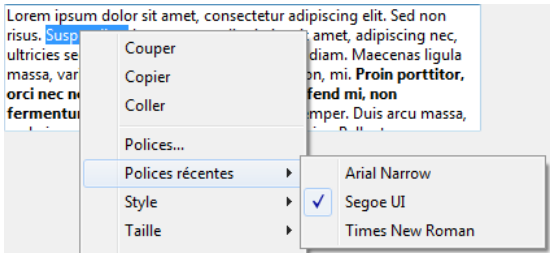
Paramètre	Type		Description
tabPolices	Tableau texte	➡	Tableau de noms de polices

Description

La commande **FIXER POLICES RECENTES** permet de modifier la liste des polices récentes affichées dans le menu contextuel des "polices récentes". Ce menu contient les noms des dernières polices sélectionnées durant la session. Il est notamment utilisé par les zones de **Notes de programmation**.

Exemple

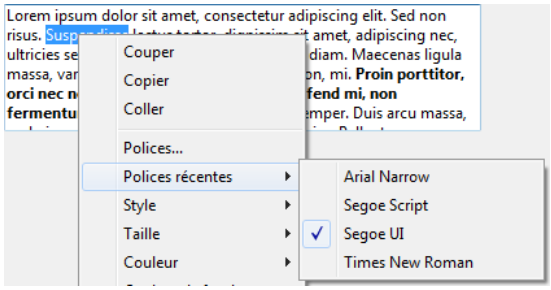
Vous souhaitez ajouter une police au menu des polices récentes :



Vous exécutez le code suivant :

```
TABLEAU TEXTE($tTRecentes;0)
LISTE DES POLICES($tTRecentes;2)
AJOUTER A TABLEAU($tTRecentes;"Segoe Script")
FIXER POLICES RECENTES($tTRecentes)
```

Le menu contient alors :



FIXER PROFONDEUR ECRAN

FIXER PROFONDEUR ECRAN (profondeur {; couleur {; écran}})

Paramètre	Type		Description
profondeur	Entier long	⇒	Profondeur de l'écran (nombre de couleurs = 2 ^ profondeur)
couleur	Entier long	⇒	1 = écran couleur 0 = écran en niveaux de gris
écran	Entier long	⇒	Numéro de l'écran ou écran principal si omis

Description

FIXER PROFONDEUR ECRAN vous permet de modifier la profondeur et les paramètres de couleurs/niveaux de gris de l'écran dont vous avez passé le numéro dans *écran*. Si vous ne passez pas ce paramètre, la commande s'applique à l'écran principal.

Pour connaître les valeurs à passer dans les paramètres *profondeur* et *couleur*, reportez-vous à la description de la commande **PROFONDEUR ECRAN**.

Gestalt

Gestalt (sélecteur ; valeur) -> Résultat

Paramètre	Type		Description
sélecteur	Chaîne	➡	Sélecteur gestalt (4 caractères)
valeur	Entier long	⬅	Résultat du gestalt
Résultat	Entier long	➲	Code d'erreur résultant

Description

La commande **Gestalt** retourne dans *valeur* une valeur numérique représentant les caractéristiques de la configuration matérielle et logicielle de votre système, en fonction du sélecteur que vous avez passé dans le paramètre *sélecteur*.

Si l'information demandée est obtenue, la fonction **Gestalt** retourne 0, sinon elle retourne l'erreur -5550. Si le sélecteur est inconnu, **Gestalt** retourne l'erreur -5551.

Important : Le Gestalt Manager est spécifique à Mac OS. Certains des sélecteurs sont également implémentés sous Windows mais l'utilité de cette fonction reste limitée sur cette plate-forme.

Pour plus d'informations sur les sélecteurs que vous pouvez passer dans **Gestalt**, reportez-vous à la documentation technique Apple relative au Gestalt Manager, consultable en ligne à l'adresse suivante : http://developer.apple.com/documentation/Carbon/Reference/Gestalt_Manager/index.html

Exemple

Dans la version 10.4.11 de Mac OS, le code suivant :

```
$vlErrCode:=Gestalt("sysv";$vlInfo)
Si($vlErrCode=0)
   ALERTE("Vous utilisez la version suivante du système : "+Chaine($vlInfo;"&x"))
Fin de si
```

... affiche l'alerte “Vous utilisez la version suivante du système : 0x1049”.

Hauteur barre de menus

Hauteur barre de menus -> Résultat

Paramètre	Type		Description
Résultat	Entier long	⮌	Hauteur (exprimée en pixels) de la barre de menus (retourne zéro si la barre de menus est cachée)

Description

La commande **Hauteur barre de menus** retourne la hauteur de la barre de menus, exprimée en pixels.
Si la barre de menus est masquée, la commande retourne 0.

Hauteur ecran

Hauteur ecran {(*)} -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Windows : hauteur de la fenêtre de l'application ou hauteur de l'écran si * est spécifié Macintosh : hauteur de l'écran principal
Résultat	Entier long	⇐	Hauteur exprimée en pixels

Description

Sous Windows, **Hauteur ecran** retourne la hauteur de la fenêtre d'application 4D (fenêtre MDI). Si vous passez le paramètre optionnel *, **Hauteur ecran** retourne la hauteur de l'écran.

Sous Mac OS, **Hauteur ecran** retourne la hauteur de l'écran principal, c'est-à-dire celui qui contient la barre de menus.

Largeur ecran

Largeur ecran {(*)} -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Windows : largeur de la fenêtre de l'application ou largeur de l'écran si * est spécifié Macintosh : largeur de l'écran principal
Résultat	Entier long	⇐	Largeur exprimée en pixels

Description

Sous Windows, **Largeur ecran** retourne la largeur de la fenêtre d'application 4D (fenêtre MDI). Si vous passez le paramètre optionnel *, **Largeur ecran** retourne la largeur de l'écran.

Sous Mac OS, **Largeur ecran** retourne la largeur de l'écran principal, c'est-à-dire celui qui contient la barre de menus.

LIRE FORMATAGE SYSTEME

LIRE FORMATAGE SYSTEME (format ; valeur)

Paramètre	Type		Description
format	Entier long	➡	Formatage système à lire
valeur	Chaîne	⬅	Valeur de formatage définie dans le système

Description

La commande **LIRE FORMATAGE SYSTEME** retourne la valeur courante de plusieurs paramètres régionaux définis dans le système d'exploitation. Cette commande permet de construire des formats personnalisés "automatiques" basés sur les préférences système.

Passez dans le paramètre *format* le type de paramètre dont vous souhaitez connaître la valeur. Le résultat est retourné directement par le système dans le paramètre *valeur* sous forme de chaîne de caractères. Vous devez passer dans *format* une des constantes du thème "**Formatages système**". Voici le descriptif de ces constantes :

Constante	Type	Valeur	Comment
Libellé AM heure système	Entier long	18	Libellé additionnel des heures d'avant midi dans les formats sur 12 heures (ex : "Matin")
Libellé PM heure système	Entier long	19	Libellé additionnel des heures après midi dans les formats sur 12 heures (ex : "Après-Midi")
Motif date abrégé	Entier long	7	Format d'affichage de date abrégé sous la forme "dddd MMMM yyyy"
Motif date court	Entier long	6	Format d'affichage de date court sous la forme "dddd MMMM yyyy"
Motif date long	Entier long	8	Format d'affichage de date long sous la forme "dddd MMMM yyyy"
Motif heure abrégé	Entier long	4	Format d'affichage d'heure abrégé sous la forme "HH:mm:ss"
Motif heure court	Entier long	3	Format d'affichage d'heure court sous la forme "HH:mm:ss"
Motif heure long	Entier long	5	Format d'affichage d'heure long sous la forme "HH:mm:ss"
Position année date courte	Entier long	17	Position de l'année dans le format de date court : "1" = à gauche, "2" = au milieu, "3" = à droite
Position jour date courte	Entier long	15	Position du jour dans le format de date court : "1" = à gauche, "2" = au milieu, "3" = à droite
Position mois date courte	Entier long	16	Position du mois dans le format de date court : "1" = à gauche, "2" = au milieu, "3" = à droite
Séparateur date	Entier long	13	Séparateur utilisé dans les formats de dates (ex : "/")
Séparateur de milliers	Entier long	1	Séparateur de milliers (ex : " ")
Séparateur décimal	Entier long	0	Séparateur décimal (ex : ",")
Séparateur heure	Entier long	14	Séparateur utilisé dans les formats d'heures (ex : ":")
Symbole monétaire	Entier long	2	Symbole monétaire (ex : "$")

Exemple

Sur un chèque rempli mécaniquement, les sommes inscrites sont généralement précédées de "*" afin d'empêcher les fraudes. Si le format d'affichage système standard pour la monnaie est "$ 5,422.33", le format pour les chèques devrait, lui, être du type "$***5432.33" : pas de virgule entre les milliers et pas d'espace entre le symbole $ et le premier chiffre. Le format à utiliser avec la fonction **Chaine** doit être "$*******.**". Pour le construire par programmation il est nécessaire de connaître le symbole monétaire et le séparateur décimal :

```
LIRE FORMATAGE SYSTEME(Symbole monétaire;$vSymbMon)
LIRE FORMATAGE SYSTEME(Séparateur décimal;$vSepDec)
$MonFormat:="###"+$vSymbMon+"*******"+$vSepDec+"**"
$Résultat:=Chaine(montant;$MonFormat)
```

LISTE DES POLICES

LISTE DES POLICES (polices {; typeListe | *})

Paramètre	Type		Description
polices	Tableau texte	⇐	Tableau des noms des polices vectorielles disponibles
typeListe | *	Entier long, Opérateur	⇒	Type de liste de police à retourner ou * pour retourner des noms de police sous OS X

Description

La commande **LISTE DES POLICES** remplit le tableau Texte *polices* avec les noms des polices vectorielles disponibles dans votre système.

Le paramètre *typeListe* vous permet de désigner le type de liste de police à obtenir. Pour cela, vous pouvez passer dans le paramètre *typeListe* l'une des constantes suivantes, placées dans le thème "**Type de liste des polices**" :

Constante	Type	Valeur	Comment
Polices favorites	Entier long	1	*polices* contient la liste des polices favorites. - Sous Windows : liste des noms de famille des polices actives. - Sous OS X : liste des noms de famille des polices présente dans le panneau de configuration nommé "Favorites" en anglais, "Favoris" en français", "Favoriten" en allemand, etc. Cette collection peut être vide si l'utilisateur n'a ajouté aucune police favorite.
Polices récentes	Entier long	2	*polices* contient la liste des polices récentes (liste des polices utilisées lors de la session 4D). Cette liste est notamment utilisée par les zones de texte multistyle.
Polices système	Entier long	0	*polices* contient la liste de toutes les polices système. Option par défaut si *typeListe* est omis.

Si vous passez le paramètre optionnel *, sous OS X la commande remplira le tableau *polices* avec les noms des polices elles-mêmes et non avec les noms des familles de police. Le fonctionnement par défaut simplifie la gestion programmée des zones de texte multistyle, qui utilisent des familles de police. Si vous passez le paramètre *, les noms de police, par exemple "Arial bold", "Arial italic", "Arial narrow italic", seront retournés au lieu des familles "Arial", "Arial black" ou "Arial narrow".
Sous Windows, le paramètre * n'a aucun effet. La commande retourne toujours les familles de police.

Note : Sous OS X, si vous utilisez le résultat de cette commande avec la commande **ST FIXER ATTRIBUTS**, il est impératif de ne pas passer le paramètre *.

A propos des polices vectorielles

Cette commande ne retourne que les polices vectorielles. En effet, l'utilisation de polices non vectorielles (i.e. polices bitmap) pour dessiner des interfaces est déconseillé car elles sont basées sur une technologie dépassée et souffrent de limitations quant aux variations de taille. Elles ne sont pas prises en charge dans les fonctionnalités les plus récentes de 4D telles que les zones 4D Write Pro.

Sous OS X, ce principe est appliqué depuis OS X 10.4 (les polices bitmap *QuickDraw* sont obsolètes à compter de cette version).

Sous Windows, ce principe est appliqué à compter de 4D v15 R4 afin d'aider les développeurs 4D à ne sélectionner que des polices modernes pour leurs interfaces. Seules les polices vectorielles "trueType" ou "openType" sont listées. Par exemple, "ASI_Mono", "MS Sans Serif" ou encore "System" ne sont pas proposées. De plus, les noms GDI sont également ignorés ; seuls les noms de familles de police DirectWrite sont pris en charge. Par exemple, les familles "Arial Black" ou "Segoe UI Black" ne sont pas dans la liste ; seuls "Arial" et "Segoe" sont retournés.

Notes de compatibilité Windows :

- Les polices bitmap peuvent toujours être utilisées dans vos formulaires 4D (à l'exception des zones 4D Write Pro). Elles sont uniquement supprimées de la liste retournée par cette commande. Cependant, pour assurer la compatibilité de vos applications avec les versions futures de 4D et de Windows, nous recommandons dès à présent d'utiliser uniquement les familles de police DirectWrite.
- Comme les polices bitmap sont filtrées dans le paramètre *polices* sous Windows, la liste résultante est différente dans les applications 4D v15 R4 et suivantes par rapport aux versions précédentes. Pensez à adapter votre code si vous utilisiez cette commande pour sélectionner une police non vectorielle.

Exemple 1

Dans un formulaire, vous voulez obtenir une liste déroulante qui affiche les polices disponibles dans le système. Ecrivez la méthode suivante pour votre objet liste déroulante :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TABLEAU TEXTE(taPolices;0)
      LISTE DES POLICES(taPolices)
   ` ...
Fin de cas
```

Exemple 2

Vous souhaitez obtenir la liste des polices récentes :

```
LISTE DES POLICES($tabPolices;Polices récentes)
```

LISTE STYLES POLICE

LISTE STYLES POLICE (famillePolice ; listeStylesPolice ; listeNomsPolice)

Paramètre	Type		Description
famillePolice	Texte	➡	Nom de la famille de police
listeStylesPolice	Tableau texte	⬅	Liste des styles pris en charge par la famille de police
listeNomsPolice	Tableau texte	⬅	Liste des noms complets pris en charge par la famille de police

Description

La commande **LISTE STYLES POLICE** retourne la liste des styles et la liste des noms complets pris en charge par la famille de police désignée par le paramètre *famillePolice*. Cette commande vous permet de concevoir des interfaces manipulant les familles de polices et les styles de police, en particulier dans le contexte des zones **4D Write Pro**.

Dans *famillePolice*, passez le nom de la famille de police dont vous souhaitez connaître les styles et les noms complets.

Dans *listeStylesPolice*, passez un tableau texte qui sera rempli avec la liste des styles pris en charge par la *famillePolice*. Les styles sont retournés avec leurs noms localisés (i.e. un élément "Italique" sera "Itálico" sur un système espagnol), ce qui vous permet par exemple de construire dynamiquement un pop-up menu "Styles" localisé.

Dans *listeNomsPolice*, passez un tableau texte qui sera rempli avec la liste complète des noms de police pris en charge par la *famillePolice*. A la différence du tableau *listeStylesPolice*, le tableau *listeNomsPolice* retourne des valeurs non localisées, i.e. des noms de police basés sur leur identifiant système. Ainsi, les noms de police seront indépendants de la langue du système. Les éléments de ce tableau sont des chaînes de caractères destinées à être utilisées avec l'attribut wk font de la commande 4D Write Pro **WP FIXER ATTRIBUTS**. Grâce à cette fonctionnalité, les documents 4D Write Pro stockent les noms de police et peuvent donc être ouverts sur différentes machines, quelle que soit la langue du système, sans problèmes de polices.

Si la *famillePolice* n'est pas trouvée sur la machine, les tableaux sont retournés vides. Pour connaître la liste des familles de police disponibles sur la machine, utilisez la commande **LISTE DES POLICES**.

Exemple

Vous voulez sélectionner les styles de la famille de police "Verdana" (si elle est disponible) :

```
TABLEAU TEXTE($aTfonts;0)
TABLEAU TEXTE($aTStyles;0)
TABLEAU TEXTE($aTnames;0)
C_ENTIER LONG($numStyle)

LISTE DES POLICES($aTfonts)
$numStyle:=Chercher dans tableau($aTfonts;"Verdana")
Si($numStyle#0)
   LISTE STYLES POLICE($aTfont{$numStyle};$aTStyles;$aTnames)
Fin de si

 //Par exemple, les tableaux résultants sont :
 //$aTStyles{1}="Normal"
 //$aTStyles{1}="Italique"
 //$aTStyles{1}="Gras"
 //$aTStyles{1}="Gras Italique"

 // $aTnames{1}="Verdana"
 // $aTnames{1}="Verdana Italic"
 // $aTnames{1}="Verdana Bold"
 // $aTnames{1}="Verdana Bold Italic"
```

Nom de la machine

Nom de la machine -> Résultat

Paramètre	Type	Description
Résultat	Chaîne	Nom de la machine sur le réseau

Description

La commande **Nom de la machine** retourne le nom de la machine tel qu'il a été défini dans les paramètres réseau du système d'exploitation.

Exemple

Même si vous n'utilisez pas la version client/serveur de 4D, votre application peut comprendre des services réseaux qui nécessitent que votre machine soit correctement configurée. Dans la **Méthode base Sur ouverture** de votre application, vous pouvez écrire :

```
Si((Nom de la machine="")|(Nom du possesseur=""))
  ` Afficher une boîte de dialogue demandant à l'utilisateur de configurer ses paramètres réseau
Fin de si
```

Nom du possesseur

Nom du possesseur -> Résultat

Paramètre	Type		Description
Résultat	Chaîne		Nom du possesseur de la machine sur le réseau

Description

La fonction **Nom du possesseur** retourne le nom du possesseur de la machine, tel qu'il a été défini dans le compte d'utilisateur courant sur la machine.

Exemple

Reportez-vous à l'exemple de la commande **Nom de la machine**.

Nombre ecrans

Nombre ecrans -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Nombre d'écrans

Description

Nombre ecrans retourne le nombre de moniteurs qui sont connectés à votre machine.

OUVRIR SELECTEUR COULEUR

OUVRIR SELECTEUR COULEUR {(texteOuFond)}

Paramètre	Type		Description
texteOuFond	Entier long	⇨	0 ou omis = couleur du texte, 1 = couleur du fond du texte

Description

La commande **OUVRIR SELECTEUR COULEUR** affiche la boîte de dialogue de sélection de couleur du système.

Note : Cette boîte de dialogue est modale sous Windows mais pas sous OS X.

Si l'utilisateur sélectionne une couleur et valide la boîte de dialogue, la couleur choisie est appliquée à la sélection courante de texte dans l'objet ayant le focus, si la propriété "Autoriser sélecteur couleur/police" est cochée pour cet objet (cf. manuel *Mode Développement*).

Si vous passez 0 dans le paramètre *texteOuFond* ou omettez ce paramètre, la couleur sélectionnée sera appliquée au texte. Si vous passez 1 dans *texteOuFond*, la couleur sélectionnée sera appliquée au fond du texte.

Si la couleur a été modifiée, l'événement formulaire Sur après modification est généré pour l'objet.

OUVRIR SELECTEUR POLICE

OUVRIR SELECTEUR POLICE
Ne requiert pas de paramètre

Description

La commande **OUVRIR SELECTEUR POLICE** affiche la boîte de dialogue de sélection de police du système.

Note : Cette boîte de dialogue est modale sous Windows mais pas sous OS X.

Si l'utilisateur sélectionne une police et/ou un style et valide la boîte de dialogue, les modifications sont appliquées à la sélection courante de texte dans l'objet ayant le focus, si la propriété "Autoriser sélecteur couleur/police" est cochée pour cet objet (cf. manuel *Mode Développement*). Dans le cas contraire, la commande ne fait rien.

Si la police a été modifiée, l'événement formulaire Sur après modification est généré pour l'objet.

Exemple

Dans un formulaire, vous souhaitez ajouter un bouton affichant le sélecteur de police afin de permettre à l'utilisateur de modifier la police ou le style d'une zone de variable texte. Assurez-vous que :

- la variable texte dispose de la propriété "Autoriser sélecteur couleur/police".
- la propriété "Focusable" est désélectionnée pour le bouton.

Le code du bouton est le suivant :

```
Au cas ou
   :(Evenement formulaire=Sur clic)
      ALLER A OBJET(textVar) //donner le focus à la variable
      OUVRIR SELECTEUR POLICE
Fin de cas
```

PROFONDEUR ECRAN

PROFONDEUR ECRAN (profondeur ; couleur {; écran})

Paramètre	Type		Description
profondeur	Entier long	⇐	Profondeur de l'écran (nombre de couleurs = 2 ^ profondeur)
couleur	Entier long	⇐	1 = écran couleur 0 = écran noir et blanc ou niveaux de gris
écran	Entier long	⇒	Numéro de l'écran ou écran principal si omis

Description

La commande **PROFONDEUR ECRAN** retourne dans les paramètres *profondeur* et *couleur* les caractéristiques du moniteur utilisé.

La profondeur de l'écran est retournée dans *profondeur*. La profondeur élevée en tant que puissance de 2 vous permet de connaître le nombre de couleurs que votre moniteur affiche. Si par exemple votre moniteur est paramétré en 256 couleurs (2^8), la profondeur de votre écran est de 8. 4D fournit les constantes prédéfinies suivantes :

Constante	Type	Valeur
Deux cent cinquante six coul	Entier long	8
Milliers de couleurs	Entier long	16
Millions de couleurs 24 bits	Entier long	24
Millions de couleurs 32 bits	Entier long	32
Noir et blanc	Entier long	0
Quatre couleurs	Entier long	2
Seize couleurs	Entier long	4

Si le moniteur est configuré pour afficher des couleurs, le paramètre *couleur* vaut 1. Si le moniteur est configuré pour afficher des niveaux de gris, *couleur* vaut 0 (zéro). Notez que cette valeur n'a de signification que sous Mac OS. Les constantes prédéfinies suivantes sont fournies par 4D :

Constante	Type	Valeur
Est en couleurs	Entier long	1
Est en niveaux de gris	Entier long	0

Le paramètre optionnel *écran* vous permet de spécifier le numéro du moniteur sur lequel vous souhaitez obtenir des informations. Si vous omettez ce paramètre, la commande retourne la profondeur de l'écran principal.

Exemple

Votre application affiche de nombreux graphiques en couleurs. Vous pouvez écrire, quelque part dans votre base :

```
PROFONDEUR ECRAN($vlProf;$vlCouleur)
Si($vlProf<16)
   ALERTE("Les formulaires seraient plus beaux si l'écran"+" était configuré en millions de couleurs.")
Fin de si
```

PROPRIETES PLATE FORME

PROPRIETES PLATE FORME (plateForme {; système {; processeur {; langue}}})

Paramètre	Type		Description
plateForme	Entier long	⇦	2 = Mac OS, 3 = Windows
système	Entier long	⇦	Dépend de la version que vous utilisez
processeur	Entier long	⇦	Famille de processeur
langue	Entier long	⇦	Dépend du système que vous utilisez

Description

La commande **PROPRIETES PLATE FORME** retourne des informations sur le type de système d'exploitation que vous utilisez, la version et la langue du système d'exploitation ainsi que le processeur installé.
PROPRIETES PLATE FORME retourne ces informations dans les paramètres *plateForme*, *système*, *processeur* et *langue*.

plateForme indique le système d'exploitation utilisé. Ce paramètre retourne une des constantes prédéfinies suivantes :

Constante	Type	Valeur
Mac OS	Entier long	2
Windows	Entier long	3

Les informations retournées dans le paramètre *système* dépendent de la version de 4D que vous utilisez.

Version Macintosh

Avec une version Mac OS de 4D, le paramètre *système* retourne une valeur sur 32 bits (Entier long), dans laquelle le "mot machine haut" est inutilisé et le "mot machine bas" est structuré ainsi :

- L'octet supérieur contient le numéro de version principal,
- L'octet inférieur est composé de deux "nibbles" de 4 bits chacun. Le nibble supérieur est le numéro de mise à jour principal et le nibble inférieur le numéro de mise à jour secondaire. Par exemple : le système 9.0.4 est codé $0904, vous recevrez donc la valeur décimale 2308.

Note : Dans 4D, vous pouvez extraire ces valeurs à l'aide des **Opérateurs numériques** *%* (modulo) et \ (division entière) ou des **Opérateurs sur les bits**.

Utilisez la formule suivante pour connaître le numéro de version principal de Mac OS :

```
PROPRIETES PLATE FORME($vlPlatform;$vlSystem)
$vlResult:=$vlSystem\256
 //Si $vlResult = 16 --> vous êtes sous Mac OS 10.x
 //Si $vlResult # 16 --> vous êtes sous une autre version de Mac OS
```

Version Windows

Avec une version Windows de 4D, le paramètre *système* retourne une valeur sur 32 bits (Entier long), structurée ainsi :

Si le bit supérieur vaut 0, vous utilisez Windows NT, Windows 2000, Windows XP ou Windows Vista. S'il vaut 1, vous utilisez une version trop ancienne de Windows.

Note : Le bit supérieur détermine le signe de la valeur Entier long. De ce fait, dans 4D, vous avez simplement besoin de tester la valeur retournée par *système* ; si elle est négative, vous utilisez une version obsolète de Windows. Vous pouvez également utiliser les **Opérateurs sur les bits**.

L'octet inférieur fournit le numéro de version principal de Windows :

- S'il vaut 4, vous utilisez Windows NT 4. S'il vaut 5, vous utilisez Windows 2000, Windows Server 2003 ou Windows XP (le signe de la valeur indique si vous utilisez Windows NT/2000 ou non).
- S'il vaut 6, vous utilisez Windows Vista, Windows Seven ou Windows 8.1.
- S'il vaut 10, vous utilisez Windows 10. A noter que dans ce cas, le paramètre *système* vaut également 10.

L'octet inférieur suivant fournit le numéro de version secondaire de Windows.

Note : Dans 4D, vous pouvez extraire ces valeurs à l'aide des **Opérateurs numériques** *%* (modulo) et \ (division entière) ou des **Opérateurs sur les bits**.
Le paramètre *processeur* indique la "famille" du microprocesseur de la machine. Deux valeurs peuvent être renvoyées, disponibles sous forme de constantes :

Constante	Type	Valeur
Compatible Intel	Entier long	586
Power PC	Entier long	406

La combinaison des paramètres *plateForme* et *processeur* permet par exemple de savoir sans ambiguïté si la machine utilisée est de type “MacIntel” (*plateForme=Mac OS* et *processeur=Compatible Intel*).

Le paramètre *langue* permet de connaître la langue courante du système sur lequel est exécutée la base. Voici la liste des codes pouvant être retournés dans ce paramètre, ainsi que leur signification :

Code	Langue
1	Arabe
2	Bulgare
3	Catalan
4	Chinois
5	Tcheque
6	Danois
7	Allemand
8	Grec
9	Anglais
10	Espagnol
11	Finlandais
12	Français
13	Hébreu
14	Hongrois
15	Islandais
16	Italien
17	Japonais
18	Coréen
19	Néerlandais
20	Norvégien
21	Polonais
22	Portugais
24	Roumain
25	Russe
26	Croate
26	Serbe
27	Slovaque
28	Albanais
29	Suédois
30	Thailandais
31	Turc
33	Indonésien
34	Ukrainien
35	Bélarusse
36	Slovène
37	Estonien
38	Letton
39	Lithuanien
41	Farsi
42	Vietnamien
45	Basque
54	Afrikaans
56	Féroïen

Note : Si la commande n'a pas pu identifier la langue du système, la valeur 9 (Anglais) est retournée.

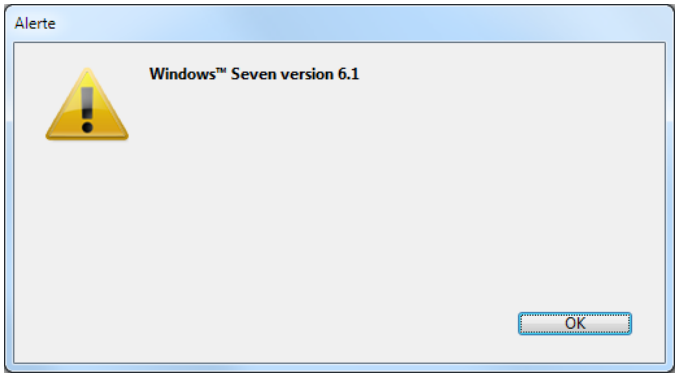
Exemple

La méthode projet suivante affiche une boîte de dialogue d'alerte décrivant le système d'exploitation utilisé :

```
 // Méthode projet VERSION SYSTEME
PROPRIETES PLATE FORME($vlPlatform;$vlSystem;$vlMachine)
Si(($vlPlatform<2)|($vlPlatform>3))
   $vsPlatformOS:=""
Sinon
   Si($vlPlatform=Windows)
      $vsPlatformOS:=""
      Si($vlSystem<0)
         $vsPlatformOS:="Version Windows trop ancienne"
      Sinon
         $winMajVers:=$vlSystem%256
         $winMinVers:=($vlSystem\256)%256
         Au cas ou
            :($winMajVers=4)
               $vsPlatformOS:="Windows™ NT"
            :($winMajVers=5)
               Au cas ou
                  :($winMinVers=0)
                     $vsPlatformOS:="Windows™ 2000"
                  :($winMinVers=1)
                     $vsPlatformOS:="Windows™ XP"
                  :($winMinVers=2)
```

```
                        $vsPlatformOS:="Windows™ 2003"
                     Sinon
                        $vsPlatformOS:="Windows (version indéterminée)"
                  Fin de cas
               :($winMajVers=6)
                  Au cas ou
                     :($winMinVers=0)
                        $vsPlatformOS:="Windows™ Vista"
                     :($winMinVers=1)
                        $vsPlatformOS:="Windows™ Seven"
                     :($winMinVers=3)
                        $vsPlatformOS:="Windows™ 8.1"
                     Sinon
                        $vsPlatformOS:="Windows (version indéterminée)"
                  Fin de cas
               :($winMajVers=10) //$vlSystem=10 également
                  $vsPlatformOS:="Windows™ 10"
            Fin de cas
         Fin de si
         $vsPlatformOS:=$vsPlatformOS+" version "+Chaine($winMajVers)+"."+Chaine($winMinVers)
      Sinon
         $vsPlatformOS:="OS X version "
         Si(($vlSystem\256)=16)
            $vsPlatformOS:=$vsPlatformOS+"10"
         Sinon
            $vsPlatformOS:=$vsPlatformOS+Chaine($vlSystem\256)
         Fin de si
         $vsPlatformOS:=$vsPlatformOS+"."+Chaine(($vlSystem\16)%16)+
(("."+Chaine($vlSystem%16))*Num(($vlSystem%16)#0))
      Fin de si
   Fin de si
   ALERTE($vsPlatformOS)
```

Sous Windows, vous obtenez une boîte de dialogue semblable à celle-ci :



Sous Mac OS, vous obtenez une boîte de dialogue semblable à celle-ci :

Selectionner couleur RVB

Selectionner couleur RVB {(coulDefaut {; message})} -> Résultat

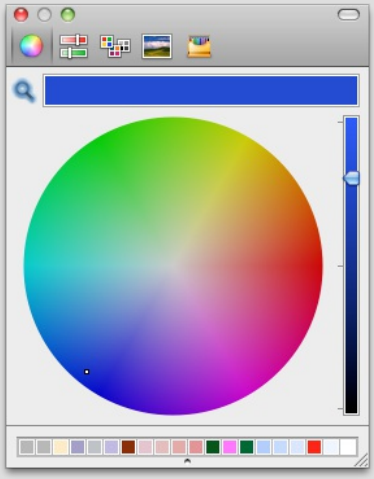
Paramètre	Type		Description
coulDefaut	Entier long	⇒	Couleur RVB présélectionnée
message	Alpha	⇒	Titre de la fenêtre de sélection
Résultat	Entier long	⇐	Couleur RVB

Description

La commande **Selectionner couleur RVB** affiche la fenêtre système de sélection de couleur et retourne la valeur RVB de la couleur sélectionnée par l'utilisateur.

La fenêtre système de sélection de couleur a l'apparence suivante :

Macintosh



Windows

Le paramètre facultatif *coulDéfaut* vous permet de pré-sélectionner une couleur dans la fenêtre. Ce paramètre vous permet par exemple de restituer par défaut la dernière couleur définie par l'utilisateur. Passez dans ce paramètre une valeur de couleur au format RVB (pour plus d'informations, reportez-vous à la description de la commande **OBJET FIXER COULEURS RVB**). Vous pouvez utiliser l'une des constantes du thème **FIXER COULEUR RVB**. Si le paramètre *coulDéfaut* est omis ou si vous passez 0, la couleur noir est sélectionnée à l'ouverture de la boîte de dialogue.

Le paramètre facultatif *message* vous permet de personnaliser le titre de la fenêtre système. Par défaut, si ce paramètre est omis, le libellé "Couleurs" est affiché.

La prise en compte de la validation de la boîte de dialogue diffère selon la plate-forme :

- Sous Windows, si l'utilisateur clique sur le bouton **OK**, la commande retourne la valeur de couleur sélectionnée au format RVB et la variable système *OK* prend la valeur 1. Si l'utilisateur annule la boîte de dialogue, la commande retourne -1 et la variable système *OK* prend la valeur 0.
- Sous Mac OS, la boîte de dialogue peut uniquement être refermée via un clic sur la case de fermeture ou en appuyant sur la touche **Echap**. Dans les deux cas, la variable système *OK* prend la valeur 1, quelles que soient les actions utilisateur dans la fenêtre. La commande retourne la valeur de couleur sélectionnée au format RVB. Si l'utilisateur n'a pas sélectionné de couleur, la valeur retournée est la valeur éventuellement passée dans *coulDefaut* ou 0 si vous n'avez pas passé ce paramètre.

Note : Cette commande ne doit pas être exécutée sur le poste serveur ni dans le cadre d'un process Web.

_o_Nom de police

_o_Nom de police (numéro) -> Résultat

Paramètre	Type		Description
numéro	Entier long	➡	Numéro de la police de laquelle récupérer le nom
Résultat	Chaîne	⮌	Nom de la police

Note de compatibilité

Cette commande est obsolète et ne doit plus être utilisée à compter de 4D v14. Elle est conservée pour des raisons de compatibilité mais ne sera plus prise en charge dans les prochaines versions du programme.

_o_Numéro de police

_o_Numéro de police (policeNom) -> Résultat

Paramètre	Type		Description
policeNom	Chaîne	➡	Nom de la police de laquelle récupérer le numéro
Résultat	Entier long	⮌	Numéro de police

Note de compatibilité

Cette commande est obsolète et ne doit plus être utilisée à compter de 4D v14. Elle est conservée pour des raisons de compatibilité mais ne sera plus prise en charge dans les prochaines versions du programme.

Etats rapides

- QR APPELER SUR COMMANDE
- QR BLOB VERS ETAT
- QR Chercher colonne
- QR CREER ZONE
- QR Creer zone hors ecran
- QR DEPLACER COLONNE
- QR ETAT
- QR ETAT VERS BLOB
- QR EXECUTER
- QR EXECUTER COMMANDE
- QR FIXER DESTINATION
- QR FIXER DONNEES TOTAUX
- QR FIXER ENCADREMENTS
- QR FIXER ENTETE ET PIED DE PAGE
- QR FIXER ESPACEMENT TOTAUX
- QR FIXER INFO COLONNE
- QR FIXER INFO LIGNE
- QR FIXER MODELE HTML
- QR FIXER PROPRIETE DOCUMENT
- QR FIXER PROPRIETE TEXTE
- QR FIXER PROPRIETE ZONE
- QR FIXER SELECTION
- QR FIXER TABLE ETAT
- QR FIXER TRIS
- QR FIXER TYPE ETAT
- QR INSERER COLONNE
- QR Lire colonne deposee
- QR LIRE DESTINATION
- QR LIRE DONNEES TOTAUX
- QR LIRE ENCADREMENTS
- QR LIRE ENTETE ET PIED DE PAGE
- QR LIRE ESPACEMENT TOTAUX
- QR LIRE INFO COLONNE
- QR Lire info ligne
- QR Lire modele HTML
- QR Lire propriete document
- QR Lire propriete texte
- QR Lire propriete zone
- QR LIRE SELECTION
- QR Lire statut commande
- QR Lire table etat
- QR LIRE TRIS
- QR Lire type etat
- QR Nombre de colonnes
- QR SUPPRIMER COLONNE
- QR SUPPRIMER ZONE HORS ECRAN

QR APPELER SUR COMMANDE

QR APPELER SUR COMMANDE (zone ; nomMéthode)

Paramètre	Type		Description
zone	Entier long	➡	Référence de la zone
nomMéthode	Chaîne	➡	Nom de la méthode à appeler

Description

La commande **QR APPELER SUR COMMANDE** exécute la méthode projet 4D dont le nom est passé dans le paramètre *nomMéthode* lorsqu'une commande de l'éditeur d'états rapides est appelée via la sélection d'un menu ou le clic sur un bouton.

Si le paramètre *zone* vaut 0 (zéro), la méthode *nomMéthode* sera appelée pour toutes les zones de l'éditeur d'états rapides jusqu'à ce que la base soit refermée ou que l'instruction suivante soit exécutée : **QR APPELER SUR COMMANDE(0;"")**.

La méthode *nomMéthode* reçoit deux paramètres :

- *$1* contient la référence de la zone (Entier long).
- *$2* contient le numéro de la commande sélectionnée (Entier long). Vous pouvez comparer cette valeur aux constantes du thème **QR Commandes**.

Note : Si vous souhaitez compiler votre base à l'aide du **Compilateur**, vous devez déclarer explicitement les paramètres $1 et $2 en entiers longs, même si vous ne les utilisez pas.

Si vous souhaitez que la commande initiale choisie par l'utilisateur soit exécutée, utilisez l'instruction suivante dans la méthode *nomMéthode* : **QR EXECUTER COMMANDE($1;$2)**.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR BLOB VERS ETAT

QR BLOB VERS ETAT (zone ; blob)

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
blob	BLOB	⇨	BLOB contenant l'état

Description

La commande **QR BLOB VERS ETAT** place l'état contenu dans le paramètre *blob* dans la zone d'état rapide désignée par le paramètre *zone*.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *blob* est incorrect, l'erreur -9852 est générée.

Exemple 1

Le code suivant affiche dans la zone MaZone l'état rapide "etat.4qr", stocké à côté du fichier de structure de la base. A noter que le fichier d'état peut avoir été créé avec une version de 4D antérieure à la 2003 :

```
C_BLOB($doc)
C_ENTIER LONG(MaZone)
DOCUMENT VERS BLOB("etat.4qr";$doc)
QR BLOB VERS ETAT(MaZone;$doc)
```

Exemple 2

L'instruction suivante affiche l'état stocké dans le champ ChampBlob dans la zone MaZone :

```
QR BLOB VERS ETAT(MaZone;[Table 1]ChampBlob)
```

QR Chercher colonne

QR Chercher colonne (zone ; expression) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
expression	Chaîne, Pointeur	⇒	Objet de colonne
Résultat	Entier long	⇐	Numéro de colonne

Description

La commande **QR Chercher colonne** retourne le numéro de la première colonne de la *zone* dont le contenu correspond à l'*expression* passée en paramètre.

expression peut contenir soit une chaîne soit un pointeur.

QR Chercher colonne retourne -1 si la recherche n'aboutit pas.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

Exemple

Le code suivant permet de récupérer le numéro de la colonne contenant le champ [G.ER Tests]Quarter puis de supprimer la colonne :

```
$NumColonne:=QR Chercher colonne(MaZone;->[G.ER Tests]Quarter)
```

ou :

```
$NumColonne:=QR Chercher colonne(MaZone;"[G.ER Tests]Quarter")
```

suivi de :

```
Si($NumColonne#-1)
   QR SUPPRIMER COLONNE(MaZone;$NumColonne)
Fin de si
```

QR CREER ZONE

QR CREER ZONE (ptr)

Paramètre	Type		Description
ptr	Pointeur	⇨	Pointeur vers une variable entier long

Description

La commande **QR CREER ZONE** crée une nouvelle zone d'état rapide et stocke son numéro de référence dans la variable de type Entier long référencée par le pointeur *ptr*.

QR Creer zone hors ecran

QR Creer zone hors ecran -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Référence de la zone créée

Description

La commande **QR Creer zone hors ecran** crée une zone d'Etat rapide hors écran et retourne son numéro de référence.

QR DEPLACER COLONNE

QR DEPLACER COLONNE (zone ; numColonne ; nouvPosition)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
numColonne	Entier long	⇒	Numéro de la colonne
nouvPosition	Entier long	⇒	Nouvelle position de la colonne

Description

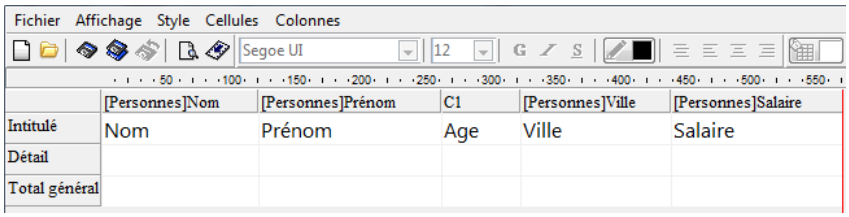
La commande **QR DEPLACER COLONNE** déplace la colonne *numColonne* de sa position courante à la position *nouvPosition*.

Les deux paramètres *numColonne* et *nouvPosition* doivent être des numéros de colonne valides (entre 1 et le nombre total de colonnes de l'état) ; dans le cas contraire, l'erreur -9852 est retournée.

Note : Cette commande peut être utilisée avec des états en liste uniquement.

Exemple

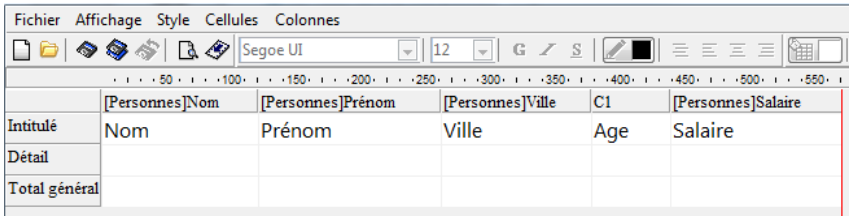
Vous avez conçu l'état suivant :



Si vous exécutez :

```
QR DEPLACER COLONNE(area;3;4)
```

Vous obtenez :



QR ETAT

QR ETAT ({laTable ;} nomFichier {; hiérarchique {; assistant {; recherche {; nomMéthode {; *}}}}})

Paramètre	Type		Description
laTable	Table	⇒	Table à utiliser ou Table par défaut si ce paramètre est omis
nomFichier	Chaîne	⇒	Document d'état rapide à charger
hiérarchique	Booléen	⇒	(versions 32 bits uniquement) Vrai = Afficher les tables N liées Faux ou omis = Ne pas les afficher
assistant	Booléen	⇒	(versions 32 bits uniquement) Vrai = Afficher le bouton de l'assistant Faux ou omis = Ne pas l'afficher
recherche	Booléen	⇒	(versions 32 bits uniquement) Vrai = Afficher les outils de recherche et la table principale; Faux ou omis = Ne pas les afficher
nomMéthode	Chaîne	⇒	(versions 32 bits uniquement) Nom de la méthode à appeler
*	Opérateur	⇒	Suppression des boîtes de dialogue d'impression

Description

La commande **QR ETAT** imprime un état pour *laTable*, à l'aide de l'Editeur d'états rapides de 4D. Cet éditeur permet à l'utilisateur de construire en totalité son propre état. Pour plus d'informations sur la création d'états à l'aide de l'Editeur d'états rapides, reportez-vous à la section **Etats rapides** ou **Etats rapides (64 bits)** dans le manuel *Mode Développement* de 4D.

Notes

- L'éditeur n'apparaît pas si la *laTable* a été déclarée "Invisible".
- Lorsque l'éditeur est appelé via la commande **QR ETAT**, les liens entre les tables conservent leur statut manuel, le cas échéant. Ce principe permet au développeur de gérer lui-même ce statut à l'aide des commandes **FIXER LIENS AUTOMATIQUES** et **FIXER LIEN CHAMP**. En version 32 bits, l'option **Tous les liens en automatique**, permettant de modifier le statut automatique/manuel des liens, est masquée.
- L'éditeur est appelé dans une fenêtre externe, il n'est pas possible d'utiliser la commande **QR APPELER SUR COMMANDE** dans ce contexte. Vous pouvez cependant utiliser le paramètre *nomMéthode* afin d'exécuter du code personnalisé lorsqu'une commande d'interface est activée (cf. ci-dessous).

Le paramètre *nomFichier* désigne un modèle d'état créé dans l'éditeur d'états rapides et sauvegardé sur disque. Le document stocke les paramètres de l'état, pas les enregistrements. Si une chaîne vide ("") est passée dans *nomFichier*, **QR ETAT** affiche une boîte de dialogue d'ouverture de fichiers, dans laquelle l'utilisateur peut choisir un modèle d'état à imprimer.
Si le paramètre *nomFichier* spécifie un document qui n'existe pas (si vous passez, par exemple, **Caractere(1)** dans *nomFichier*), l'éditeur d'états rapides s'affiche.

Versions 32 bits uniquement :

- Le paramètre *hiérarchique* indique si les tables liées N doivent être ou non affichées dans la liste de sélection de champs. Par défaut, sa valeur est 0 (les tables N ne sont pas affichées).
- Le paramètre *assistant* permet d'indiquer si le bouton **Ouvrir l'assistant** doit apparaître ou non dans la fenêtre de l'éditeur d'états rapides. Passez Vrai pour afficher le bouton et Faux pour le masquer. Par défaut (si ce paramètre est omis), le bouton n'est pas affiché.
- Le paramètre *recherche* permet d'indiquer si le bouton **Nouvelle recherche** doit apparaître ou non dans la fenêtre de l'éditeur d'états rapides. Passez Vrai pour afficher le bouton et Faux pour le masquer. Par défaut (si ce paramètre est omis), le bouton n'est pas affiché.
- Le paramètre *nomMéthode* permet de désigner une méthode projet 4D qui sera exécutée à chaque fois qu'une commande de l'éditeur d'états rapides sera appelée via la sélection d'un menu ou un clic sur un bouton. Utiliser ce paramètre équivaut à utiliser la commande **QR APPELER SUR COMMANDE** dans le contexte de la fenêtre de l'Editeur d'états rapides (**QR APPELER SUR COMMANDE** fonctionne uniquement dans le contexte d'une zone incluse). Ce paramètre permet notamment de modifier le jeu de caractères utilisé par l'état rapide.
  La méthode *nomMéthode* reçoit deux paramètres :
  - *$1* contient la référence de la zone (Entier long).
  - *$2* contient le numéro de la commande sélectionnée (Entier long). Vous pouvez comparer cette valeur aux constantes du thème [#title id="249"/].

  Note : Si vous souhaitez compiler votre base à l'aide du Compilateur, vous devez déclarer explicitement les paramètres $1 et $2 en entiers longs, même si vous ne les utilisez pas.

  Si vous souhaitez que la commande initiale choisie par l'utilisateur soit exécutée, utilisez l'instruction suivante dans la méthode *nomMéthode* :

```
QR EXECUTER COMMANDE($1;$2)
```

  Si le paramètre *nomMéthode* est une chaîne vide ("") ou est omis, aucune méthode ne sera appelée et le fonctionnement standard de **QR ETAT** s'appliquera.

Une fois qu'un fichier d'état est sélectionné, les boîtes de dialogue d'impression s'affichent, sauf si le paramètre * a été spécifié — dans ce cas, elles ne s'affichent pas. L'état est alors imprimé.

Lorsque l'Editeur d'états rapides n'est pas affiché, la variable système OK prend la valeur 1 si un état est imprimé ; sinon elle prend la valeur 0 (zéro) — par exemple si l'utilisateur a cliqué sur **Annuler** dans les boîtes de dialogue d'impression.

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :

- Veillez à ce qu'aucune boîte de dialogue n'apparaisse sur le poste serveur (sauf besoin spécifique). Pour cela, il est nécessaire d'appeler la commande avec le paramètre *.
- La syntaxe faisant apparaître l'éditeur d'états rapide ne fonctionne pas avec 4D Server, dans ce cas la variable système OK prend la valeur 0.
- En cas de problème sur l'imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Exemple 1

L'exemple suivant permet à l'utilisateur d'effectuer une recherche dans la table [Personnes], puis imprime automatiquement l'état "Liste détaillée" :

```
CHERCHER([Personnes])
Si(OK=1)
   QR ETAT([Personnes];"Liste détaillée";Faux;Faux;Faux;*)
```

```
Fin de si
```

Exemple 2

L'exemple suivant permet à l'utilisateur d'effectuer une recherche dans la table [Personnes], puis de sélectionner le document d'état qui sera ensuite utilisé pour l'impression :

```
CHERCHER([Personnes])
Si(OK=1)
   QR ETAT([Personnes];"";Faux;Faux;Faux)
Fin de si
```

Exemple 3

L'exemple suivant permet à l'utilisateur d'effectuer une recherche dans la table [Personnes], puis affiche l'Editeur d'états rapides afin que l'utilisateur puisse construire, charger, sauvegarder ou imprimer tout état, avec ou sans l'assistant :

```
CHERCHER([Personnes])
Si(OK=1)
   QR ETAT([Personnes];Caractere(1);Faux;Vrai)
Fin de si
```

Exemple 4

Reportez-vous à l'exemple de la commande **FIXER LIEN CHAMP**.

Exemple 5

Vous souhaitez convertir le jeu de caractères utilisé dans un état rapide appelé via **QR ETAT** en Mac Roman :

```
QR ETAT([MaTable];Caractere(1);Faux;Faux;Faux;"maCallbackMeth")
```

La méthode ***maCallbackMeth*** convertit l'état lorsqu'il est généré :

```
C_ENTIER LONG($1;$2)
Si($2=qr cmd executer) //si on a généré un état
   C_BLOB($myblob)
   C_TEXTE($path;$text)
   C_ENTIER LONG($type)
   QR EXECUTER COMMANDE($1;$2) //exécution de la commande
   QR LIRE DESTINATION($1;$type;$path) //récupération de la destination
   Si(($type=qr fichier HTML)|$type=qr fichier texte))
      DOCUMENT VERS BLOB($path;$myblob)
 //conversion vers un texte en utilisant UTF-8
      $text:=Convertir vers texte($myblob;"UTF-8")
 //utilisation du jeu MacRoman
      CONVERTIR DEPUIS TEXTE($text;"MacRoman";$myblob)
 //Renvoi de l'état converti
      BLOB VERS DOCUMENT($path;$myblob)
   Fin de si
Sinon //sinon exécution de la commande
   QR EXECUTER COMMANDE($1;$2)
Fin de si
```

QR ETAT VERS BLOB

QR ETAT VERS BLOB (zone ; blob)

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
blob	BLOB	⇦	BLOB devant recevoir l'état rapide

Description

La commande **QR ETAT VERS BLOB** place dans le BLOB *blob* (variable ou champ) l'état dont la référence a été passée dans le paramètre *zone*.
Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

Exemple

L'instruction suivante affecte l'état rapide stocké dans la zone MaZone à un champ BLOB :

```
QR ETAT VERS BLOB(MaZone;[Table 1]ChampBlob)
```

QR EXECUTER

QR EXECUTER (zone)

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone à exécuter

Description

La commande **QR EXECUTER** provoque l'exécution de l'état rapide désigné par le paramètre *zone*. L'état est généré avec ses paramétrages courants, notamment son type de sortie. Vous pouvez utiliser la commande **QR FIXER DESTINATION** pour modifier le type de sortie.

L'état est exécuté sur la table à laquelle appartient la zone. Lorsque *zone* désigne une zone hors écran, il est nécessaire de spécifier la table à utiliser à l'aide de la commande **QR FIXER TABLE ETAT**.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte, veillez à ce qu'aucune boîte de dialogue n'apparaisse sur le poste serveur (sauf besoin spécifique). Pour cela, il est nécessaire d'appeler la commande **QR FIXER DESTINATION** avec le paramètre "*". En cas de problème sur l'imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

QR EXECUTER COMMANDE

QR EXECUTER COMMANDE (zone ; numCommande)

Paramètre	Type		Description
zone	Entier long	➡	Référence de la zone
numCommande	Entier long	➡	Commande de menu à exécuter

Description

La commande **QR EXECUTER COMMANDE** exécute la commande de menu ou le bouton de la barre d'outils dont la référence est passée dans le paramètre *numCommande*. En général, cette commande est utilisée pour exécuter une commande de menu sélectionnée par l'utilisateur et interceptée dans votre code via la commande **QR APPELER SUR COMMANDE**.

Passez dans le paramètre *numCommande* une des constantes du thème **QR Commandes** :

Constante	Type	Valeur
qr cmd ajouter colonne	Entier long	2608
qr cmd aligné à droite	Entier long	505
qr cmd aligné à gauche	Entier long	503
qr cmd aligné par défaut	Entier long	512
qr cmd apercu	Entier long	2007
qr cmd cacher colonne	Entier long	2602
qr cmd cacher ligne	Entier long	2607
qr cmd centré	Entier long	504
qr cmd déplacer à droite	Entier long	3003
qr cmd déplacer à gauche	Entier long	3002
qr cmd destination 4D View	Entier long	2503
qr cmd destination fichier	Entier long	2501
qr cmd destination fichier HTML	Entier long	2504
qr cmd destination graphe	Entier long	2502
qr cmd destination imprimante	Entier long	2500
qr cmd écart type	Entier long	513
qr cmd éditer colonne	Entier long	2603
qr cmd encadrements	Entier long	2609
qr cmd enregistrer	Entier long	2002
qr cmd enregistrer sous	Entier long	2003
qr cmd entete et pied de page	Entier long	2005
qr cmd espacement des totaux	Entier long	2610
qr cmd executer	Entier long	2008
qr cmd format	Entier long	2606
qr cmd gras	Entier long	500
qr cmd insérer colonne	Entier long	2600
qr cmd italique	Entier long	501
qr cmd largeur automatique	Entier long	2605
qr cmd liste déroulante police	Entier long	1000
qr cmd max	Entier long	509
qr cmd min	Entier long	508
qr cmd mise en forme	Entier long	2611
qr cmd mise en page	Entier long	2006
qr cmd moyenne	Entier long	507
qr cmd nombre	Entier long	510
qr cmd normal	Entier long	511
qr cmd nouveau	Entier long	2000
qr cmd objet couleur fond	Entier long	1003
qr cmd objet couleur fond alt	Entier long	1004
qr cmd objet couleur texte	Entier long	1002
qr cmd ouvrir	Entier long	2001
qr cmd palette colonnes	Entier long	2054
qr cmd palette couleurs de fond	Entier long	2052
qr cmd palette opérateurs	Entier long	2051
qr cmd palette standard	Entier long	2053
qr cmd palette style	Entier long	2050
qr cmd somme	Entier long	506
qr cmd souligné	Entier long	502
qr cmd supprimer colonne	Entier long	2601
qr cmd valeurs répétées	Entier long	2604
qr cmd version enregistrée	Entier long	2004

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numCommande* est incorrect, l'erreur -9852 est générée.

QR FIXER DESTINATION

QR FIXER DESTINATION (zone ; type {; spécificités})

Paramètre	Type		Description
zone	Entier long	➡	Référence de la zone
type	Entier long	➡	Type d'état
spécificités	Chaîne, Variable	➡	Spécificités du type de destination

Description

La commande **QR FIXER DESTINATION** permet de définir le *type* de destination de sortie de l'état rapide contenu dans la *zone*.

Passez dans le paramètre *type* une des constantes du thème **QR Destination de sortie**. Le contenu du paramètre *spécificités* dépend de la valeur de *type*. Le tableau suivant liste les valeurs qui peuvent être passées dans les paramètres *type* et *spécificités*.

Constante	Type	Valeur	Comment
_o_qr zone 4D Chart	Entier long	4	*** Constante obsolète ***
qr fichier HTML	Entier long	5	*spécificités* : Chemin d'accès du fichier.
qr fichier texte	Entier long	2	*spécificités* : Chemin d'accès du fichier.
qr imprimante	Entier long	1	*spécificités* : "*" pour supprimer les boîtes de dialogue d'impression
qr zone 4D View	Entier long	3	*spécificités* : N.A.

qr imprimante (1) : Si vous passez une chaîne contenant une étoile ("*") dans le paramètre *spécificités*, aucune boîte de dialogue ne sera affichée lors de l'impression, les paramètres d'impression courants seront automatiquement utilisés. Ce paramétrage est nécessaire si vous souhaitez imprimer l'état sur le serveur.

qr fichier texte (2) : Si vous passez une chaîne vide dans le paramètre *spécificités*, une boîte de dialogue standard d'enregistrement de fichiers apparaît. Si vous passez un chemin d'accès valide, l'état rapide sera enregistré à l'emplacement indiqué.
Par défaut, le délimiteur de champ est le caractère Tabulation (code 9) et le délimiteur d'enregistrement est le caractère Retour chariot (code 13). Vous pouvez modifier ces caractères par défaut en changeant la valeur des variables système FldDelimit et RecDelimit. Sous Windows, si FldDelimit vaut 13, un caractère 10 (Saut de ligne) sera ajouté après le Retour chariot. Tenez compte du fait que ces variables sont utilisées par d'autres commandes, par exemple **IMPORTER TEXTE**. Toute modification de ces variables est répercutée sur l'ensemble de l'application.

qr zone 4D View(3) : Si l'utilisateur courant dispose du plug-in 4D View, une fenêtre externe 4D View est créée et affiche les résultats des paramètres courants de la zone d'état rapide.

qr fichier HTML (5) : Un fichier HTML est généré d'après les paramètres courants de la zone d'état rapide. Le fichier HTML est basé sur le modèle défini par la commande **QR FIXER MODELE HTML**. Pour plus d'informations sur le mode de conversion des données, veuillez vous référer au manuel Mode Développement.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si la valeur de *type* de destination est incorrecte, l'erreur -9852 est générée.

Exemple

L'exemple suivant définit le fichier texte "MonDoc.txt" comme type de destination de l'état puis l'exécute :

```
QR FIXER DESTINATION(MaZone;qr fichier texte;"MonDoc.txt")
QR EXECUTER(MaZone)
```

QR FIXER DONNEES TOTAUX

QR FIXER DONNEES TOTAUX (zone ; numColonne ; numRupture ; opérateur | valeur)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
numColonne	Entier long	⇒	Numéro de colonne
numRupture	Entier long	⇒	Numéro de rupture
opérateur | valeur	Entier long, Chaîne	⇒	Opérateur pour la cellule ou Contenu de la cellule

Description

Note : Cette commande ne crée pas de sous-total.

Etat en liste

La commande **QR FIXER DONNEES TOTAUX** permet de définir le contenu d'une ligne de rupture spécifique (sous-total ou total général).

Passez dans *zone* la référence de la zone d'état rapide.

Passez dans *numColonne* le numéro de colonne de la cellule que vous souhaitez définir.

Passez dans *numRupture* le numéro de la ligne de rupture à modifier (sous-total ou total général). Pour une ligne de sous-total, *numRupture* correspond au numéro d'ordre de la rupture. Pour le total général, *numRupture* vaut -3 (vous pouvez également utiliser la constante *qr total général* du thème **QR Lignes pour Propriétés**).

Le paramètre *opérateur* contient la valeur cumulée de tous les opérateurs à placer dans la cellule. Utilisez les constantes du thème **QR Opérateurs** pour définir ce paramètre :

Constante	Type	Valeur
qr écart type	Entier long	32
qr max	Entier long	8
qr min	Entier long	4
qr moyenne	Entier long	2
qr nombre	Entier long	16
qr somme	Entier long	1

Si vous ne souhaitez utiliser aucun opérateur, passez 0 dans le paramètre *opérateur*.

Si vous choisissez d'insérer du texte dans la cellule, passez-le dans le paramètre *valeur*.

Note : Les paramètres *opérateur* et *valeur* sont mutuellement exclusifs, vous pouvez passer soit une combinaison de valeurs numériques, soit du texte.

Si vous souhaitez saisir à la fois du texte et des opérateurs, vous pouvez utiliser les codes suivants dans le paramètre *valeur* :
- # pour la valeur provoquant la rupture ou le sous-total
- ##S sera remplacé par la somme.
- ##A sera remplacé par la moyenne.
- ##C sera remplacé par le nombre
- ##X sera remplacé par le maximum.
- ##N sera remplacé par le minimum.
- ##D sera remplacé par l'écart type.
- ##xx, où xx est un numéro de colonne. Ce code sera remplacé par la valeur de la colonne désignée, dans son propre formatage. Si la colonne n'existe pas, le code apparaît dans l'état.

Etat tableau croisé

La commande **QR FIXER DONNEES TOTAUX** vous permet de définir le contenu d'une cellule spécifique.

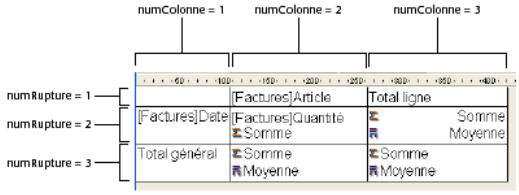
Passez dans *zone* la référence de la zone d'état rapide.

Passez dans *numColonne* le numéro de colonne de la cellule que vous souhaitez définir.

Passez dans *numRupture* le numéro de ligne de la cellule que vous souhaitez définir.

Le paramètre *opérateur* contient la valeur cumulée de tous les opérateurs à placer dans la cellule. Utilisez les constantes du thème **QR Opérateurs** pour définir ce paramètre (cf. paragraphe précédent).
Le paramètre alternatif *valeur* permet de définir le texte à insérer dans la cellule.

L'illustration suivante précise la manière dont les paramètres *numColonne* et *numRupture* sont combinés dans un tableau croisé :



Types de données acceptés

Vous pouvez passer deux types de données : des libellés et des opérateurs.

- Libellés
  Un libellé est une chaîne de caractères passée via le paramètre *valeur*. Cette valeur ne peut être utilisée qu'avec les cellules suivantes : *numColonne=3,numRupture=1* et *numColonne=1,numRupture=3*.
- Opérateurs
  Un opérateur ou un cumul d'opérateurs (cf. paragraphe précédent) peut être passé via le paramètre *opérateur* aux cellules suivantes : *numColonne=2,numRupture=2*

numColonne=3,numRupture=2
numColonne=2,numRupture=3
Notez que ces deux dernières valeurs affectent également la cellule (colonne 3,ligne 3). En effet, si par exemple un calcul est effectué dans la cellule (colonne 2,ligne 3), le contenu de la cellule (colonne 3/ligne 3) sera modifié en conséquence.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numColonne* est incorrect, l'erreur -9852 est générée.
Si le paramètre *numRupture* est incorrect, l'erreur -9853 est générée.

QR FIXER ENCADREMENTS

QR FIXER ENCADREMENTS (zone ; colonne ; ligne ; encadrement ; ligne {; couleur})

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
colonne	Entier long	⇒	Numéro de colonne
ligne	Entier long	⇒	Numéro de ligne
encadrement	Entier long	⇒	Valeur d'encadrements composée
ligne	Entier long	⇒	Epaisseur de ligne
couleur	Entier long	⇒	Couleur de ligne

Description

La commande **QR FIXER ENCADREMENTS** permet de définir le style d'encadrement d'une cellule spécifique.

Passez dans *zone* la référence de la zone d'état rapide.

Passez dans *colonne* le numéro de colonne de la cellule à encadrer.

Le paramètre *ligne* contient le numéro de ligne de la cellule à encadrer. Vous pouvez passer soit :

- une valeur entière positive pour désigner la ligne de sous-total correspondante,
- l'une des constantes suivantes, placées dans le thème **QR Lignes pour Propriétés** :

Constante	Type	Valeur	Comment
qr détail	Entier long	-2	Zone Détail de l'état
qr intitulé	Entier long	-1	Intitulé de l'état
qr total général	Entier long	-3	Zone Total général

Le paramètre *encadrement* contient une valeur composée permettant d'indiquer la ou les bordures(s) de cellule à modifier. Passez l'une des constantes du thème **QR Encadrements** :

Constante	Type	Valeur	Comment
qr encadrement bas	Entier long	8	Bordure inférieure
qr encadrement droit	Entier long	4	Bordure droite
qr encadrement gauche	Entier long	1	Bordure gauche
qr encadrement haut	Entier long	2	Bordure supérieure
qr encadrement horiz intérieur	Entier long	32	Bordure intérieure horizontale
qr encadrement vert intérieur	Entier long	16	Bordure intérieure verticale

Le paramètre *encadrement* peut contenir un cumul de plusieurs valeurs afin de désigner simultanément plusieurs bordures. Par exemple, si vous passez 5 dans *encadrement*, les bordures droite et gauche seront affectées.

Le paramètre *épaisseur* permet de spécifier l'épaisseur de l'encadrement :

- 0 indique une épaisseur nulle (pas de bordure),
- 1 indique une épaisseur d'1/4 point,
- 2 indique une épaisseur d'1/2 point,
- 3 indique une épaisseur d'1 point,
- 4 indique une épaisseur de 2 points.

Le paramètre *couleur* permet de désigner la couleur de l'encadrement :

- si vous passez une valeur positive, *couleur* désigne un numéro de couleur,
- si vous passez 0, la couleur est noire,
- si vous passez -1, la couleur esrt inchangée.

Note : Par défaut, la couleur est noire.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *colonne* est incorrect, l'erreur -9852 est générée.
Si le paramètre *ligne* est incorrect, l'erreur -9853 est générée.
Si le paramètre *encadrement* est incorrect, l'erreur -9854 est générée.
Si le paramètre *épaisseur* est incorrect, l'erreur -9855 est générée.

QR FIXER ENTETE ET PIED DE PAGE

QR FIXER ENTETE ET PIED DE PAGE (zone ; sélecteur ; titreGauche ; titreCentre ; titreDroit ; hauteur {; image {; alignementImage}})

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
sélecteur	Entier long	⇒	1 = En-tête, 2 = Pied de page
titreGauche	Chaîne	⇒	Texte affiché sur le côté gauche
titreCentre	Chaîne	⇒	Texte affiché au centre
titreDroit	Chaîne	⇒	Texte affiché sur le côté droit
hauteur	Entier long	⇒	Hauteur de l'en-tête ou du pied de page
image	Image	⇒	Image à afficher
alignementImage	Entier long	⇒	Alignement de l'image

Description

La commande **QR FIXER ENTETE ET PIED DE PAGE** vous permet de définir le contenu et la taille de l'en-tête et du pied de page de la *zone*.

Le paramètre *sélecteur* vous permet de définir la zone à affecter :
• si *sélecteur* vaut 1, l'en-tête sera affecté ;
• si *sélecteur* vaut 2, le pied de page sera affecté.

Les paramètres *titreGauche*, *titreCentre* et *titreDroit* définissent le contenu des en-têtes ou pieds de page situés respectivement à gauche, au centre et à droite.

Le paramètre *hauteur* indique la hauteur de l'en-tête ou du pied de page, exprimée dans l'unité courante de l'état rapide.

Le paramètre *image* contient l'image à afficher facultativement dans l'en-tête ou le pied de page.

Le paramètre *alignementImage* définit la propriété d'alignement de l'image passée dans *image* :

- si *alignementImage* vaut 1, l'image est alignée sur la gauche.
- si *alignementImage* vaut 2, l'image est centrée.
- si *alignementImage* vaut 3, l'image est alignée sur la droite.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *sélecteur* est incorrect, l'erreur -9852 est générée.

Exemple

L'instruction suivante place le libellé "Titre du centre" dans l'en-tête de l'état rapide dans MaZone et définit sa hauteur à 200 points :

```
QR FIXER ENTETE ET PIED DE PAGE(MaZone;1;"";"Titre du centre";"";200)
```

QR FIXER ESPACEMENT TOTAUX

QR FIXER ESPACEMENT TOTAUX (zone ; sousTotal ; valeur)

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
sousTotal	Entier long	⇨	Numéro de sous-total
valeur	Entier long	⇨	0=pas d'espace, 32000=insère un saut de page, >0=espace ajouté sous le niveau de rupture, <0=augmentation proportionnelle

Description

La commande **QR FIXER ESPACEMENT TOTAUX** permet de définir l'espacement ajouté au-dessous d'une ligne de sous-total. Elle ne peut être utilisée qu'avec des états en liste.

Le paramètre *zone* contient la référence de la zone d'état rapide.
Le paramètre *sousTotal* désigne le niveau de sous-total (ou de rupture) à modifier.

Le paramètre *valeur* permet de définir la valeur de l'espacement :

- Si *valeur* vaut 0, aucun espacement n'est ajouté.
- Si *valeur* vaut 32000, un saut de page est ajouté.
- Si *valeur* est une valeur positive, elle exprime l'espacement à ajouter en pixels.
- Si *valeur* est une valeur négative, elle exprime l'espacement à ajouter en pourcentage de la ligne de sous-total. Par exemple, la valeur -100 définit l'ajout d'un espace au-dessous de la ligne de sous-total correspondant à 100% de la hauteur courante de la ligne.

Note : Si l'espacement ajouté au-dessous d'une ligne de sous-total “repousse” la ligne suivante sur la page suivante, aucun espace n'apparaîtra au-dessus de la ligne sur cette page.

Si un numéro de *zone* invalide est passé, l’erreur -9850 est générée.
Si le paramètre *sousTotal* est incorrect, l’erreur -9852 est générée.

QR FIXER INFO COLONNE

QR FIXER INFO COLONNE (zone ; numColonne ; titre ; objet ; cachée ; taille ; valeursRépétées ; formatAffich)

Paramètre	Type		Description
zone	Entier long	➡	Référence de la zone
numColonne	Entier long	➡	Numéro de colonne
titre	Chaîne	➡	Titre de la colonne
objet	Champ, Variable	➡	Objet affecté à la colonne
cachée	Entier long	➡	0 = visible, 1 = invisible
taille	Entier long	➡	Largeur de la colonne
valeursRépétées	Entier long	➡	0 = Non répétées, 1 = Répétées
formatAffich	Chaîne	➡	Format d'affichage

Description

Etats en liste

La commande **QR FIXER INFO COLONNE** vous permet de définir les paramètres d'une colonne existante de l'état présent dans la *zone*.

Passez dans *zone* la référence de la zone d'état rapide.

Passez dans *numColonne* le numéro de la colonne à définir.

Passez dans *titre* l'intitulé devant apparaître dans l'en-tête de la colonne.

Passez dans *objet* la référence de l'objet devant être affecté à la colonne (variable, champ ou formule).

Le paramètre *cachée* indique si la colonne doit être affichée ou masquée :

- si *cachée* vaut 1, la colonne est masquée ;
- si *cachée* vaut 0, la colonne est affichée.

Passez dans *taille* la taille en pixels à assigner à la colonne. Si *taille* vaut -1, la taille de la colonne est automatique.

valeursRépétées indique le statut de la propriété de répétition des données. Par exemple, si la valeur d'un champ ou d'une variable ne change pas d'un enregistrement à l'autre, il est possible de la répéter ou non dans chaque ligne de la colonne.

- si *valeursRépétées* vaut 0, les valeurs ne sont pas répétées.
- si *valeursRépétées* vaut 1, les valeurs sont répétées.

Le paramètre *formatAffich* indique le format d'affichage de la colonne. Vous pouvez utiliser tout format d'affichage standard de 4D compatible avec les données affichées dans la colonne.

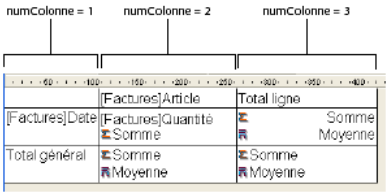
Exemple :
La ligne suivante associe à la colonne 1 l'intitulé “Titre” et le champ Champ2, rend la colonne visible avec une largeur de 150 pixels et définit le format d'affichage ###,##.

```
QR FIXER INFO COLONNE(zone;1;"Titre";"[Table 1]Champ2";0;150;0;"###,##")
```

Etats tableaux croisés

Avec ce type d'état, la commande **QR FIXER INFO COLONNE** permet de définir globalement les mêmes paramètres que ceux décrits dans le paragraphe ci-dessus, toutefois les zones auxquelles ils s'appliquent sont différentes et varient en fonction du paramètre à fixer.
En outre, les paramètres *titre*, *cachée* et *valeursRépétées* ne sont pas utilisés lorsque vous travaillez avec des états tableaux croisés. La valeur à passer dans le paramètre *numColonne* dépend de l'opération que vous souhaitez effectuer : définir la taille de la colonne ou définir la source de données et le format d'affichage.

- Taille de la colonne
  Il s'agit d'un attribut “visuel”, par conséquent les colonnes sont simplement numérotées de gauche à droite, comme illustré ci-dessous :

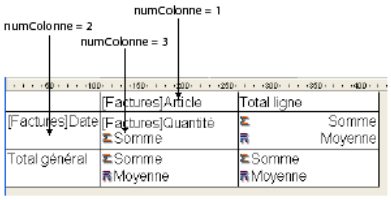


La méthode suivante affecte une taille automatique à toutes les colonnes d'un état en tableau croisé et laisse les autres éléments inchangés :

```
Boucle($i;1;3)
   QR LIRE INFO COLONNE(qr_zone;$i;$titre;$obj;$caché;$taille;$rep;$format)
   QR FIXER INFO COLONNE(qr_zone;$i;$titre;$obj;$caché;0;$rep;$format)
Fin de boucle
```

A noter que, comme vous voulez modifier uniquement la taille de la colonne, vous devez utiliser la commande **QR LIRE INFO COLONNE** pour récupérer les propriétés courantes de la colonne puis les passer à **QR FIXER INFO COLONNE** afin de les conserver inchangées, excepté pour la taille.

- Source de données (objet) et format d'affichage
  Dans ce cas, la numérotation des colonnes s'effectue de la manière suivante :

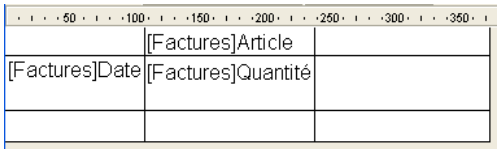


A noter qu'il n'est pas possible d'adresser toutes les cellules avec la commande **QR FIXER INFO COLONNE**, les cellules non numérotées dans le schéma ci-dessus doivent être gérées à l'aide de la commande **QR FIXER DONNEES TOTAUX**.

Le code suivant associe des sources de données aux trois cellules nécessaires à la construction d'un état en tableau croisé simple :

```
QR FIXER TABLE ETAT(qr_zone;Table(->[Factures]))
TOUT SELECTIONNER([Factures])
QR FIXER TYPE ETAT(qr_zone;2)
QR FIXER INFO COLONNE(qr_zone;1;"";->[Factures]Article;1;-1;1;"")
QR FIXER INFO COLONNE(qr_zone;2;"";->[Factures]Date;1;-1;1;"")
QR FIXER INFO COLONNE(qr_zone;3;"";->[Factures]Quantité;1;-1;1;"")
```

La zone d'état suivante est générée :



Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numColonne* est incorrect, l'erreur -9852 est générée.

QR FIXER INFO LIGNE

QR FIXER INFO LIGNE (zone ; ligne ; cachée)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
ligne	Entier long	⇒	Ligne
cachée	Entier long	⇒	0 = Visible, 1 = Cachée

Description

La commande **QR FIXER INFO LIGNE** permet d'afficher ou de masquer la ligne dont la référence est passée dans le paramètre *ligne*.

Le paramètre *ligne* désigne la ligne à modifier. Vous pouvez passer soit :

- une valeur entière positive pour désigner la ligne de sous-total correspondante,
- une des constantes suivantes du thème **QR Lignes pour Propriétés** :

Constante	Type	Valeur	Comment
qr détail	Entier long	-2	Zone Détail de l'état
qr intitulé	Entier long	-1	Intitulé de l'état
qr total général	Entier long	-3	Zone Total général

cachée permet de spécifier si le contenu de la ligne doit être affiché ou masqué :

- si *cachée* vaut 1, le contenu de la ligne est masqué ;
- si *cachée* vaut 0, le contenu de la ligne est affiché.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *ligne* est incorrect, l'erreur -9852 est générée.

Exemple

L'instruction suivante masque le contenu de la ligne Détail :

```
QR FIXER INFO LIGNE(maZone;qr détail;1)
```

QR FIXER MODELE HTML

QR FIXER MODELE HTML (zone ; modèle)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
modèle	Texte	⇒	Code du modèle HTML

Description

La commande **QR FIXER MODELE HTML** permet de définir le *modèle* HTML à utiliser pour la zone d'état rapide référencée par *zone*. Ce modèle sera utilisé lors de l'exécution des états au format HTML.

Le modèle est construit à l'aide d'un ensemble de balises de traitement des données. Ce fonctionnement vous permet de générer des documents HTML proches des états originaux ou des documents à l'apparence entièrement personnalisée.

Note : Vous devez appeler au préalable **QR FIXER DESTINATION** pour définir le format HTML comme destination de sortie.

Balises HTML

<!--#4DQRheader--> ... <!--/#4DQRheader-->
Les intitulés des colonnes seront insérés entre ces balises. Ces balises sont généralement utilisées pour définir la ligne de titre de l'état.

<!--#4DQRrow--> ... <!--/#4DQRrow-->
Les informations insérées entre ces balises seront répétées pour chaque ligne de données (détail et sous-total compris).

<!--#4DQRcol--> ... <!--/#4DQRcol-->
Les informations insérées entre ces balises seront répétées pour chaque colonne de données à l'intérieur des lignes. Le tri de la colonne est identique à celui de l'état. Lorsqu'elles sont utilisées conjointement à *<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->*, les balises *<!--#4DQRcol--> ... <!--/#4DQRcol-->* ne seront effectives qu'avec les colonnes dont le contenu n'est pas inséré à l'aide de *<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->*.
Par exemple, dans un état comportant cinq colonnes, vous utilisez les balises *<!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->* afin d'insérer les données de la deuxième colonne. *<!--#4DQRcol--> ... <!--/#4DQRcol-->* traiteront, pour chaque ligne, les colonnes 1, 3, 4 et 5. Ces balises ignoreront la colonne dont le contenu est publié à l'aide de *<!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->*.

<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->
Les informations insérées entre ces balises seront extraites de la colonne de l'état dont le numéro est “n”. Si, par exemple, dans un état HTML à trois colonnes, vous souhaitez afficher les colonnes dans un ordre différent de celui de l'état initial, vous pouvez écrire :
<!--#4DQRrow--> <!--#4DQRcol;3--> ... <!--/#4DQRcol;3--><!--#4DQRcol;2--> ... <!--/#4DQRcol;2--><!--#4DQRcol;1--> ... <!--/#4DQRcol;1--> <!--/#4DQRrow-->
Dans cet exemple, les colonnes sont générées dans l'ordre inverse de l'état.

<!--#4DQRfont--> ... <!--/#4DQRfont-->
Les informations insérées entre ces balises seront utilisées pour la définition de la police de la colonne ou cellule courante.
<!--#4DQRfont--> sera remplacé par une définition de police HTML et *<!--/#4DQRfont-->* sera remplacé par la balise de fermeture standard (**).

<!--#4DQRface--> ... <!--/#4DQRface-->
Les informations insérées entre ces balises seront utilisées pour la définition du style de la colonne ou cellule courante.
<!--#4DQRface--> sera remplacé par une définition de style HTML *<!--#4DQRface-->* sera remplacé par la balise de fermeture standard (*</face>*).

<!--#4DQRbgcolor-->
Cette balise de couleur sera remplacée par la définition de couleur de la cellule courante.

<!--#4DQRdata-->
Cette balise sera remplacée par les données de la cellule courante.

<!--#4DQRlHeader--><!--#4DQRdata--><!--/#4DQRlHeader-->
<!--#4DQRcHeader--><!--#4DQRdata--><!--/#4DQRcHeader-->
<!--#4DQRrHeader--><!--#4DQRdata--><!--/#4DQRrHeader-->
Ces balises seront remplacées respectivement par les données de l'en-tête gauche, central et droit.

<!--#4DQRlFooter--><!--#4DQRdata--><!--/#4DQRlFooter-->
<!--#4DQRcFooter--><!--#4DQRdata--><!--/#4DQRcFooter-->
<!--#4DQRrFooter--><!--#4DQRdata--><!--/#4DQRrFooter-->
Ces balises seront remplacées respectivement par les données du pied de page gauche, central et droit.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR FIXER PROPRIETE DOCUMENT

QR FIXER PROPRIETE DOCUMENT (zone ; propriété ; valeur)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
propriété	Entier long	⇒	1 = Dialogue d'impression, 2 = Unité du document
valeur	Entier long	⇒	Valeur de la propriété

Description

La commande **QR FIXER PROPRIETE DOCUMENT** permet d'afficher la boîte de dialogue d'impression ou de définir l'unité du document présent dans la *zone*.

Vous pouvez passer dans le paramètre *propriété* une des constantes du thème **QR Propriétés de document** :

Constante	Type	Valeur	Comment
qr dialogue d'impression	Entier long	1	Affichage de la boîte de dialogue d'impression : • Si valeur = 0, la boîte de dialogue d'impression n'est pas affichée avant l'impression. • Si valeur = 1, la boîte de dialogue d'impression est affichée avant l'impression (valeur par défaut).
qr unité	Entier long	2	Unité du document : • Si valeur = 0, l'unité du document est le point. • Si valeur = 1, l'unité du document est le centimètre. • Si valeur = 2, l'unité du document est le pouce.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si la valeur du paramètre *propriété* ou *valeur* est incorrecte, l'erreur correspondante (-9852 ou -9853) est générée,

QR FIXER PROPRIETE TEXTE

QR FIXER PROPRIETE TEXTE (zone ; numColonne ; numLigne ; propriété ; valeur)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
numColonne	Entier long	⇒	Numéro de colonne
numLigne	Entier long	⇒	Numéro de ligne
propriété	Entier long	⇒	Numéro de propriété
valeur	Entier long, Chaîne	⇒	Valeur de la propriété définie

Description

La commande **QR FIXER PROPRIETE TEXTE** permet de définir les propriétés de texte de la cellule désignée par les paramètres *numColonne* et *numLigne*.

Passez dans *zone* la référence de la zone d'état rapide.

Passez dans *numColonne* le numéro de colonne de la cellule.

Passez dans *numLigne* la référence de la ligne de la cellule. Vous pouvez passer soit :

- une valeur positive désignant la ligne de sous-total correspondante,
- une des constantes du thème **QR Lignes pour Propriétés** :

Constante	Type	Valeur	Comment
qr détail	Entier long	-2	Zone Détail de l'état
qr entête	Entier long	-4	En-tête de page
qr intitulé	Entier long	-1	Intitulé de l'état
qr pied de page	Entier long	-5	Pied de page
qr total général	Entier long	-3	Zone Total général

Note : Vous devez passer une valeur dans *numColonne* même lorsque vous passez -4 ou -5 dans le paramètre *numLigne* (dans ce cas la valeur de *numColonne* est inutilisée).

Note : Dans les tableaux croisés, le principe est similaire sauf pour les valeurs des lignes, qui sont toujours positives.

Passez dans *propriété* la valeur de la propriété de texte à modifier. Vous pouvez utiliser les constantes du thème **QR Propriétés de texte**. Dans le tableau ci-dessous, la colonne Commentaire indique les valeurs associées (paramètre *valeur*) :

Constante	Type	Valeur	Comment
_o_qr police	Entier long	1	Obsolète depuis 4D v14R3 (utiliser qr nom de police)
qr couleur de fond	Entier long	8	Numéro de couleur de fond
qr couleur de fond alternée	Entier long	9	Numéro de couleur de fond alternée
qr couleur de texte	Entier long	6	Numéro de couleur (Entier long)
qr gras	Entier long	3	Attribut gras (0 ou 1)
qr italique	Entier long	4	Attribut italique (0 ou 1)
qr justification	Entier long	7	Attribut de justification (0 = par défaut, 1 = gauche, 2 = centre et 3 = droite)
qr nom de police	Entier long	10	Nom de police tel que retourné par exemple par la commande **LISTE DES POLICES**.
qr souligné	Entier long	5	Attribut souligné (0 ou 1)
qr taille police	Entier long	2	Taille de police en points (9 à 255)

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numColonne* est incorrect, l'erreur -9852 est générée.
Si le paramètre *numLigne* est incorrect, l'erreur -9853 est générée.
Si le paramètre *propriété* est incorrect, l'erreur -9854 est générée.

Exemple

Cette méthode définit plusieurs attributs pour l'intitulé de la première colonne :

```
 //Affecte la police Times :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr nom de police;"Times")
 //Affecte la taille de police 10 points :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr taille police;10)
 //Affecte l'attribut gras :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr gras;1)
 //Affecte l'attribut italique :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr italique;1)
 //Affecte l'attribut souligné :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr souligné;1)
 //Affecte la couleur vert clair :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr couleur de texte;0x0000FF00)
```

QR FIXER PROPRIETE ZONE

QR FIXER PROPRIETE ZONE (zone ; propriété ; valeur)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
propriété	Entier long	⇒	Elément d'interface
valeur	Entier long	⇒	1 = affiché, 0 = caché

Description

La commande **QR FIXER PROPRIETE ZONE** vous permet d'afficher ou de masquer dans la *zone* l'élément d'interface (barre d'outils ou barre de menus) dont la référence est passée dans le paramètre *propriété*.

La barre de menus et les barres d'outils sont numérotées de 1 à 6 (de haut en bas) et la valeur 7 est associée au menu contextuel. Vous pouvez utiliser les constantes du thème **QR Propriétés de zone** afin de désigner l'élément d'interface :

Constante	Type	Valeur	Comment
qr barre menu	Entier long	1	Affichage de la barre de menus (Affichée=1, Cachée=0)
qr barre outils colonnes	Entier long	6	Affichage de la barre d'outils Colonnes (Affichée=1, Cachée=0)
qr barre outils couleurs	Entier long	5	Affichage de la barre d'outils Couleurs de fond (Affichée=1, Cachée=0)
qr barre outils opérateurs	Entier long	4	Affichage de la barre d'outils Opérateurs (Affichée=1, Cachée=0)
qr barre outils standard	Entier long	2	Affichage de la barre d'outils standard (Affichée=1, Cachée=0)
qr barre outils style	Entier long	3	Affichage de la barre d'outils Style (Affichée=1, Cachée=0)
qr menus contextuels	Entier long	7	Affichage des menus contextuels (Affichés=1, Cachés=0)

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *propriété* est incorrect, l'erreur -9852 est générée.

QR FIXER SELECTION

QR FIXER SELECTION (zone ; gauche ; haut {; droite {; bas}})

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
gauche	Entier long	⇒	Limite gauche
haut	Entier long	⇒	Limite supérieure
droite	Entier long	⇒	Limite droite
bas	Entier long	⇒	Limite inférieure

Description

La commande **QR FIXER SELECTION** permet de sélectionner une cellule, une ligne, une colonne ou encore la totalité de la *zone*, comme vous le feriez à l'aide de la souris. Cette commande permet également de désélectionner la sélection courante.

gauche contient le numéro de la colonne représentant la limite gauche de la sélection. Si *gauche* vaut 0, la ligne entière est sélectionnée.
haut contient le numéro de la ligne représentant la limite supérieure de la sélection. Si *haut* vaut 0, la colonne entière est sélectionnée.
droite contient le numéro de la colonne représentant la limite droite de la sélection.
bas contient le numéro de la ligne représentant la limite inférieure de la sélection.

Notes

- Si les paramètres *gauche* et *haut* valent 0, la totalité de la zone est sélectionnée.
- Pour tout désélectionner, passez -1 dans les paramètres *gauche*, *haut*, *droite* et *bas*.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR FIXER TABLE ETAT

QR FIXER TABLE ETAT (zone ; numTable)

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
numTable	Entier long	⇨	Numéro de table

Description

La commande **QR FIXER TABLE ETAT** désigne via le paramètre *numTable* le numéro de la table courante de l'état rapide dont la référence est passée dans le paramètre *zone*.

Il est impératif qu'une table soit associée à un état car l'éditeur d'états utilisera la sélection courante de cette table pour afficher les données, effectuer les calculs et propager les liens si nécessaire.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numTable* est incorrect, l'erreur -9852 est générée.

QR FIXER TRIS

QR FIXER TRIS (zone ; tabColonnes {; tabTris})

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
tabColonnes	Tableau réel	⇨	Colonnes
tabTris	Tableau réel	⇨	Ordres de tris

Description

La commande **QR FIXER TRIS** vous permet de définir l'ordre de tri de chaque colonne de l'état rapide dont la référence est passée dans *zone*.

tabColonnes : vous devez stocker dans ce tableau le numéro de chaque colonne pour laquelle vous souhaitez définir un ordre de tri.

tabTris : chaque élément de ce tableau doit contenir l'ordre de tri pour la colonne correspondante référencée dans le tableau *tabColonnes*.

- Si *tabTris{$i}* vaut 1, le tri est croissant.
- Si *tabTris{$i}* vaut -1, le tri est décroissant.

Etats tableaux croisés

Avec ce type d'état, le tableau ne peut pas comporter plus de deux éléments. Vous pouvez uniquement trier les colonnes (1) et les lignes (2). Les données (situées à l'intersection des colonnes et des lignes) ne peuvent pas être triées via cette commande.

Voici le code permettant de trier les lignes uniquement dans un état tableau croisé :

```
TABLEAU REEL($tabColonnes;1)
$tabColonnes{1}:=2
TABLEAU REEL($tabTris;1)
$tabTris{1}:=-1 `Tri décroissant des lignes
QR FIXER TRIS(qr_zone;$tabColonnes;$tabTris)
```

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR FIXER TYPE ETAT

QR FIXER TYPE ETAT (zone ; type)

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
type	Entier long	⇨	Type d'état

Description

La commande **QR FIXER TYPE ETAT** permet de définir le *type* de l'état rapide présent dans la *zone*.

Vous pouvez passer dans le paramètre *type* une des constantes du thème **QR Types d'états** :

Constante	Type	Valeur
qr état en liste	Entier long	1
qr état tableau croisé	Entier long	2

Si vous définissez à l'aide de cette commande un nouveau *type* pour un état existant, les paramétrages précédents sont supprimés et un nouvel état vide est créé.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si la valeur de *type* est incorrecte, l'erreur -9852 est générée.

QR INSERER COLONNE

QR INSERER COLONNE (zone ; numColonne ; objet)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
numColonne	Entier long	⇒	Numéro de colonne
objet	Champ, Variable, Pointeur	⇒	Objet à insérer dans la colonne

Description

La commande **QR INSERER COLONNE** insère ou crée dans *zone* une colonne à un emplacement spécifique. Les colonnes situées à droite de la colonne ajoutée seront décalées en conséquence.

numColonne indique le numéro de la colonne, correspondant à la position de la colonne — les colonnes sont numérotées de gauche à droite.

La valeur passée dans *objet* sera l'intitulé par défaut de la colonne.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

Note : Cette commande ne peut pas être utilisée avec un état en tableau croisé.

Exemple

La ligne suivante insère (ou crée) une première colonne dans la zone MaZone et la remplit avec le contenu du champ Noms. L'intitulé par défaut de la colonne sera “Noms” :

```
QR INSERER COLONNE(MaZone;1;->[Table 1]Noms)
```

QR Lire colonne deposee

QR Lire colonne deposee (zone) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
Résultat	Entier long	⇐	Emplacement du "déposer"

Description

La commande **QR Lire colonne deposee** retourne une valeur indiquant l'emplacement auquel un “déposer” a été effectué dans *zone* :

- Si la valeur est négative, elle indique un numéro de colonne (par exemple, -3 indique qu'un “déposer” a été effectué sur la colonne n° 3).
- Si la valeur est positive, elle indique que le “déposer” a été effectué sur le séparateur situé devant la colonne (par exemple, 3 indique qu'un “déposer” a été effectué après la colonne n° 2). Gardez à l'esprit qu'un “déposer” ne peut pas être effectué devant une colonne existante.

Si un numéro de *zone* invalide est passé, l’erreur -9850 est générée.

QR LIRE DESTINATION

QR LIRE DESTINATION (zone ; type {; spécificités})

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
type	Entier long	⇦	Type d'état
spécificités	Chaîne, Variable	⇦	Spécificités de la destination

Description

La commande **QR LIRE DESTINATION** retourne le *type* de destination de l'état rapide contenu dans la *zone*.

Vous pouvez comparer la valeur obtenue dans le paramètre *type* avec les constantes du thème **QR Destination de sortie**. Le tableau suivant décrit les valeurs qui peuvent être retournées dans les paramètres *type* et *spécificités* :

Constante	Type	Valeur	Comment
_o_qr zone 4D Chart	Entier long	4	*** Constante obsolète ***
qr fichier HTML	Entier long	5	*spécificités* : Chemin d'accès du fichier.
qr fichier texte	Entier long	2	*spécificités* : Chemin d'accès du fichier.
qr imprimante	Entier long	1	*spécificités* : "*" pour supprimer les boîtes de dialogue d'impression
qr zone 4D View	Entier long	3	*spécificités* : N.A.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR LIRE DONNEES TOTAUX

QR LIRE DONNEES TOTAUX (zone ; numColonne ; numRupture ; opérateur ; texte)

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
numColonne	Entier long	⇨	Numéro de colonne
numRupture	Entier long	⇨	Numéro de rupture
opérateur	Entier long	⇦	Opérateur de la cellule
texte	Chaîne	⇦	Contenu de la cellule

Description

Etat en liste

La commande **QR LIRE DONNEES TOTAUX** permet de récupérer le contenu d'une ligne de rupture spécifique (sous-total ou total général).

Passez dans *zone* la référence de la zone d'état rapide.

Passez dans *numColonne* le numéro de colonne de la cellule que vous souhaitez lire.
Passez dans *numRupture* le numéro de la ligne de rupture à lire (sous-total ou total général). Pour une ligne de sous-total, *numRupture* correspond au numéro de la ligne. Pour le total général, *numRupture* vaut -3 (vous pouvez également utiliser la constante *qr total général* du thème **QR Lignes pour Propriétés**).

Le paramètre *opérateur* retourne la valeur cumulée de tous les opérateurs éventuellement présents dans la cellule. Vous pouvez utiliser les constantes du thème **QR Opérateurs** pour traiter les valeurs retournées :

Constante	Type	Valeur
qr écart type	Entier long	32
qr max	Entier long	8
qr min	Entier long	4
qr moyenne	Entier long	2
qr nombre	Entier long	16
qr somme	Entier long	1

Si *opérateur* retourne 0, la cellule ne contient aucun opérateur.

texte retourne le texte de la cellule.

Note : Les paramètres *opérateur* et *texte* sont mutuellement exclusifs ; en fonction du contenu de la cellule, seul l'un des deux paramètres retournera une valeur.

Etat tableau croisé

La commande **QR LIRE DONNEES TOTAUX** vous permet de recupérer le contenu d'une cellule spécifique.

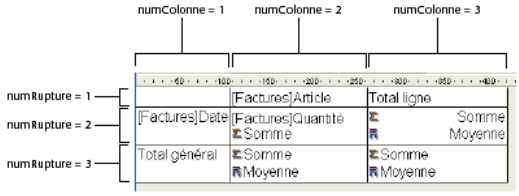
Passez dans *zone* la référence de la zone d'état rapide.

Passez dans *numColonne* le numéro de colonne et dans *numRupture* le numéro de ligne de la cellule que vous souhaitez lire.

Le paramètre *opérateur* retourne la valeur cumulée de tous les opérateurs éventuellement présents dans la cellule. Utilisez les constantes du thème **QR Opérateurs** pour évaluer la valeur récupérée (cf. paragraphe précédent).

Le paramètre *texte* retourne le contenu de la cellule.

L'illustration suivante précise la manière dont les paramètres *numColonne* et *numRupture* sont combinés dans un tableau croisé :



Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numColonne* est incorrect, l'erreur -9852 est générée.
Si le paramètre *numRupture* est incorrect, l'erreur -9853 est générée.

QR LIRE ENCADREMENTS

QR LIRE ENCADREMENTS (zone ; colonne ; ligne ; encadrement ; ligne {; couleur})

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
colonne	Entier long	⇒	Numéro de colonne
ligne	Entier long	⇒	Numéro de ligne
encadrement	Entier long	⇒	Valeur d'encadrement
ligne	Entier long	⇐	Epaisseur de trait
couleur	Entier long	⇐	Couleur de l'encadrement

Description

La commande **QR LIRE ENCADREMENTS** retourne les attributs d'encadrement d'une cellule spécifique de *zone*.

Passez dans *zone* la référence de la zone d'état rapide.

Passez dans *colonne* le numéro de colonne de la cellule à lire.

Le paramètre *ligne* contient le numéro de ligne de la cellule à lire. Vous pouvez soit :

- passer une valeur entière positive pour désigner la ligne de sous-total correspondante.
- passer une des constantes suivantes du thème **QR Lignes pour Propriétés** :

Constante	Type	Valeur	Comment
qr détail	Entier long	-2	Zone Détail de l'état
qr intitulé	Entier long	-1	Intitulé de l'état
qr total général	Entier long	-3	Zone Total général

Le paramètre *encadrement* permet d'indiquer la bordure de cellule à lire. Passez l'une des constantes du thème **QR Encadrements** :

Constante	Type	Valeur	Comment
qr encadrement bas	Entier long	8	Bordure inférieure
qr encadrement droit	Entier long	4	Bordure droite
qr encadrement gauche	Entier long	1	Bordure gauche
qr encadrement haut	Entier long	2	Bordure supérieure
qr encadrement horiz intérieur	Entier long	32	Bordure intérieure horizontale
qr encadrement vert intérieur	Entier long	16	Bordure intérieure verticale

Note : A la différence de la commande **QR FIXER ENCADREMENTS**, **QR LIRE ENCADREMENTS** n'accepte pas de valeurs cumulées. Vous devez tester séparément toutes les valeurs pour obtenir une description globale de l'encadrement de la cellule.

Le paramètre *épaisseur* retourne l'épaisseur de l'encadrement :

- 0 indique une épaisseur nulle (pas de bordure),
- 1 indique une épaisseur d'1/4 point,
- 2 indique une épaisseur d'1/2 point,
- 3 indique une épaisseur d'1 point,
- 4 indique une épaisseur de 2 points.

Le paramètre *couleur* retourne le numéro de la couleur de la bordure.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *colonne* est incorrect, l'erreur -9852 est générée.
Si le paramètre *ligne* est incorrect, l'erreur -9853 est générée.
Si le paramètre *encadrement* est incorrect, l'erreur -9854 est générée.

QR LIRE ENTETE ET PIED DE PAGE

QR LIRE ENTETE ET PIED DE PAGE (zone ; sélecteur ; titreGauche ; titreCentre ; titreDroit ; hauteur {; image {; alignementImage}})

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
sélecteur	Entier long	⇒	1 = En-tête, 2 = Pied de page
titreGauche	Chaîne	⇐	Texte affiché sur le côté gauche
titreCentre	Chaîne	⇐	Texte affiché au centre
titreDroit	Chaîne	⇐	Texte affiché sur le côté droit
hauteur	Entier long	⇐	Hauteur de l'en-tête ou du pied de page
image	Image	⇐	Image à afficher
alignementImage	Entier long	⇐	Alignement de l'image

Description

La commande **QR LIRE ENTETE ET PIED DE PAGE** vous permet de récupérer le contenu et la taille de l'en-tête et du pied de page de la *zone*.

Le paramètre *sélecteur* vous permet de définir la zone à lire :

- si *sélecteur* vaut 1, les informations de l''en-tête seront récupérées ;
- si *sélecteur* vaut 2, les informations du pied de page seront récupérées.

Les paramètres *titreGauche*, *titreCentre* et *titreDroite* retournent le contenu des en-têtes ou pieds de page situés respectivement à gauche, au centre et à droite.

Le paramètre *hauteur* retourne la hauteur de l'en-tête ou du pied de page, exprimée dans l'unité courante de l'état rapide.

Le paramètre *image* retourne le cas échéant l'image affichée dans l'en-tête ou le pied de page.
Le paramètre *alignementImage* retourne la propriété d'alignement de l'image :

- si *alignementImage* vaut 1, l'image est alignée sur la gauche.
- si *alignementImage* vaut 2, l'image est centrée.
- si *alignementImage* vaut 3, l'image est alignée sur la droite.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *sélecteur* est incorrect, l'erreur -9852 est générée.

Exemple

La méthode suivante affiche le contenu et la hauteur des libellés des en-têtes :

```
QR LIRE ENTETE ET PIED DE PAGE(MaZone;1;$TexteGauche;$TexteCentre;$TexteDroite;$Hauteur)
Au cas ou
   :($TexteGauche #"")
      ALERTE("Libellé de l'en-tête de gauche : "+Caractere(34)+$TexteGauche+Caractere(34))
   :($TexteCentre #"")
      ALERTE("Libellé de l'en-tête du centre : "+Caractere(34)+$TexteCentre+Caractere(34))
   :($TexteDroite #"")
      ALERTE("Libellé de l'en-tête de droite : "+Caractere(34)+$TexteDroite+Caractere(34))
   Sinon
      ALERTE("Aucun libellé d'en-tête dans cet Etat.")
Fin de cas
ALERTE("Hauteur des en-têtes : "+Chaine($Hauteur))
```

QR LIRE ESPACEMENT TOTAUX

QR LIRE ESPACEMENT TOTAUX (zone ; sousTotal ; valeur)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
sousTotal	Entier long	⇒	Numéro de sous-total
valeur	Entier long	⇐	0=pas d'espace, 32000=insère une saut de page, >0=espace ajouté sous le niveau de rupture, <0=augmentation proportionnelle

Description

La commande **QR LIRE ESPACEMENT TOTAUX** permet de récupérer la valeur de l'espacement ajouté au-dessous d'une ligne de sous-total. Elle ne peut être utilisée qu'avec des états en liste.

Le paramètre *zone* contient la référence de la zone d'état rapide.
Le paramètre *sousTotal* désigne le niveau de sous-total (ou de rupture) dont vous souhaitez connaître l'espacement. Ce paramètre contient une valeur comprise entre 1 et le nombre de lignes de sous-total/rupture.

Le paramètre *valeur* retourne la valeur de l'espacement :

- Si *valeur* vaut 0, aucun espacement n'est ajouté.
- Si *valeur* vaut 32000, un saut de page est ajouté.
- Si *valeur* est une valeur positive, elle exprime l'espacement en pixels.
- Si *valeur* est une valeur négative, elle exprime l'espacement en pourcentage de la ligne de sous-total. Par exemple, la valeur -100 indique un espace au-dessous de la ligne de sous-total correspondant à 100% de la hauteur courante de la ligne.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *sousTotal* est incorrect, l'erreur -9852 est générée.

QR LIRE INFO COLONNE

QR LIRE INFO COLONNE (zone ; numColonne ; titre ; objet ; cachée ; taille ; valeursRépétées ; format {; varRésultat})

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
numColonne	Entier long	⇒	Numéro de colonne
titre	Texte	⇐	Titre de la colonne
objet	Texte	⇐	Nom du champ ou contenu de la formule affecté(e) à la colonne
cachée	Entier long	⇐	0 = visible, 1 = invisible
taille	Entier long	⇐	Largeur de la colonne
valeursRépétées	Entier long	⇐	0 = non répétées, 1 = répétées
format	Texte	⇐	Format d'affichage des données
varRésultat	Texte	⇐	Nom de la variable de formule

Description

Etats en liste

La commande **QR LIRE INFO COLONNE** vous permet de récupérer les paramètres d'une colonne existante de l'état présent dans la *zone*.

Passez dans *zone* la référence de la zone d'état rapide et dans *numColonne* le numéro de la colonne à définir.

Le paramètre *titre* retourne l'intitulé de l'en-tête de la colonne.

Le paramètre *objet* retourne le nom du champ ou la formule associé(e) à la colonne.

Note : La commande ne tient pas compte de la structure virtuelle éventuellement définie via les commandes **FIXER TITRES TABLES** et **FIXER TITRES CHAMPS**. Le nom réel des champs est retourné dans le paramètre *objet*.

Le paramètre *cachée* indique si la colonne est affichée ou masquée :

- si *cachée* vaut 1, la colonne est masquée ;
- si *cachée* vaut 0, la colonne est affichée.

Le paramètre *taille* retourne la taille en pixels de la colonne. Si la valeur retournée est négative, la taille de la colonne est automatique.

valeursRépétées retourne le statut de la propriété de répétition des données. Par exemple, si la valeur d'un champ ou d'une variable ne change pas d'un enregistrement à l'autre, il est possible de la répéter ou non dans chaque ligne de la colonne.

- si *valeursRépétées* vaut 0, les valeurs ne sont pas répétées.
- si *valeursRépétées* vaut 1, les valeurs sont répétées.

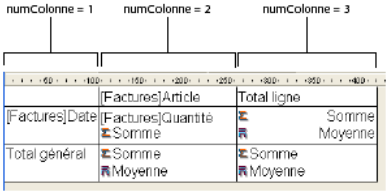
Le paramètre *format* retourne le format d'affichage de la colonne. Vous pouvez utiliser tout format d'affichage standard de 4D compatible avec les données affichées dans la colonne.

Le paramètre optionnel *varRésultat*, lorsqu'il est passé, retourne le nom de la variable automatiquement affectée par l'éditeur d'états rapides à la colonne de formule (le cas échéant) : "C1" pour la première colonne de formule, "C2" pour la seconde, et ainsi de suite. 4D utilise cette variable pour stocker les résultats de la dernière exécution de la formule de colonne lors de la génération de l'état.

Etats tableaux croisés

Avec ce type d'état, la commande **QR LIRE INFO COLONNE** permet de récupérer globalement les mêmes paramètres que ceux décrits dans le paragraphe ci-dessus, toutefois les zones auxquelles ils s'appliquent sont différentes et varient en fonction du paramètre à lire.
En outre, les paramètres *titre*, *cachée* et *valeursRépétées* ne sont pas utilisés lorsque vous travaillez avec des états tableaux croisés, les valeurs retournées dans ces paramètres ne sont donc pas significatives.
La valeur à passer dans le paramètre *numColonne* dépend de l'opération que vous souhaitez effectuer : lire la taille de la colonne ou lire la source de données et le format d'affichage.

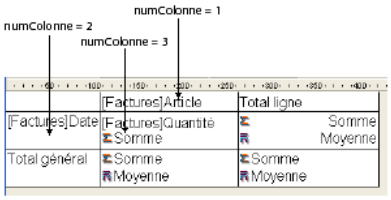
- Taille de la colonne
  Il s'agit d'un attribut “visuel”, par conséquent les colonnes sont simplement numérotées de gauche à droite, comme illustré ci-dessous :



La méthode suivante affecte une taille automatique à toutes les colonnes d'un état en tableau croisé et laisse les autres éléments inchangés :

```
Boucle($i;1;3)
   QR LIRE INFO COLONNE(qr_zone;$i;$titre;$obj;$caché;$taille;$rep;$format)
   QR FIXER INFO COLONNE(qr_zone;$i;$titre;$obj;$caché;0;$rep;$format)
Fin de boucle
```

- Source de données (objet) et format d'affichage
  Dans ce cas, la numérotation des colonnes s'effectue de la manière suivante :



Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numColonne* est incorrect, l'erreur -9852 est générée.

Exemple

Vous avez construit l'état suivant :

	[Personnes]Nom	[Personnes]Prénom	C1	[Personnes]Ville	[Personnes]Salaire
Intitulé	Nom	Prénom	Age	Ville	Salaire
Détail					
Total général					

Vous pouvez écrire :

```
C_TEXTE($vTitle;$vObject;$vDisplayFormat;$vResultVar)
C_ENTIER LONG($area;$vHide;$vSize;$vRepeatedValue)
QR LIRE INFO COLONNE($area;3;$vTitle;$vObject;$vHide;$vSize;$vRepeatedValue;$vDisplayFormat;$vResultVar)
 //$vTitle = "Age"
 //$vObject = "[Personnes]Date_Naissance-Date du jour" ou
 // "[Personnes]Date_Naissance-Current date" selon vos préférences de langage
 //$vHide = 0
 //$vSize = 57
 //$vRepeatedValue = 1
 //$vDisplayFormat = ""
 //$vResultVar = "C1"
```

QR Lire info ligne

QR Lire info ligne (zone ; ligne) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
ligne	Entier long	⇒	Ligne
Résultat	Entier long	⇐	0 = Visible, 1 = Cachée

Description

La commande **QR Lire info ligne** indique si la ligne désignée par le paramètre *ligne* est affichée ou masquée dans la *zone*.

Le paramètre *ligne* désigne la ligne à vérifier. Vous pouvez passer soit :

- une valeur entière positive pour désigner la ligne de sous-total correspondante,
- une des constantes suivantes du thème **QR Lignes pour Propriétés** :

Constante	Type	Valeur	Comment
qr détail	Entier long	-2	Zone Détail de l'état
qr intitulé	Entier long	-1	Intitulé de l'état
qr total général	Entier long	-3	Zone Total général

La valeur retournée par **QR Lire info ligne** indique si le contenu de la ligne est affiché ou masqué. Si la fonction retourne 1, le contenu de la ligne est masqué ; si elle retourne 0, le contenu de la ligne est affiché.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *ligne* est incorrect, l'erreur -9852 est générée.

QR Lire modele HTML

QR Lire modele HTML (zone) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
Résultat	Texte	⇦	Code HTML utilisé comme modèle

Description

La commande **QR Lire modele HTML** retourne le modèle HTML utilisé pour la zone d'état rapide référencée par *zone*. La valeur retournée, de type texte, contient la totalité du code HTML utilisé comme modèle.

Si aucun modèle spécifique n'a été défini, le code du modèle par défaut est retourné.
A noter que si le format de destination HTML n'a pas été défini pour l'état (manuellement ou par programmation), aucune valeur n'est retournée.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR Lire propriete document

QR Lire propriete document (zone ; propriété) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
propriété	Entier long	⇒	1=Dialogue d'impression, 2=Unité du document
Résultat	Entier long	⇐	Valeur de la propriété

Description

La commande **QR Lire propriete document** vous permet de connaître la valeur courante de la *propriété* d'affichage de la boîte de dialogue d'impression ou de l'unité du document présent dans la *zone*.

Vous pouvez passer dans le paramètre *propriété* une des constantes du thème **QR Propriétés de document** :

Constante	Type	Valeur	Comment
qr dialogue d'impression	Entier long	1	Affichage de la boîte de dialogue d'impression : • Si valeur = 0, la boîte de dialogue d'impression n'est pas affichée avant l'impression. • Si valeur = 1, la boîte de dialogue d'impression est affichée avant l'impression (valeur par défaut).
qr unité	Entier long	2	Unité du document : • Si valeur = 0, l'unité du document est le point. • Si valeur = 1, l'unité du document est le centimètre. • Si valeur = 2, l'unité du document est le pouce.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si la valeur du paramètre *propriété* est incorrecte, l'erreur -9852 est générée.

QR Lire propriete texte

QR Lire propriete texte (zone ; numColonne ; numLigne ; propriété) -> Résultat

Paramètre	Type		Description
zone	Entier long	➡	Référence de la zone
numColonne	Entier long	➡	Numéro de colonne
numLigne	Entier long	➡	Numéro de ligne
propriété	Entier long	➡	Numéro de propriété
Résultat	Chaîne, Entier long	⬅	Valeur de la propriété

Description

La commande **QR Lire propriete texte** retourne la valeur courante de la *propriété* de texte dans la cellule de *zone* désignée par *numColonne* et *numLigne*.

Passez dans *zone* la référence de la zone d'état rapide.

Passez dans *numColonne* le numéro de colonne de la cellule.

Passez dans *numLigne* la référence de la ligne de la cellule. Vous pouvez passer soit :

- une valeur positive désignant la ligne de sous-total correspondante,
- une des constantes du thème **QR Lignes pour Propriétés** :

Constante	Type	Valeur	Comment
qr détail	Entier long	-2	Zone Détail de l'état
qr entête	Entier long	-4	En-tête de page
qr intitulé	Entier long	-1	Intitulé de l'état
qr pied de page	Entier long	-5	Pied de page
qr total général	Entier long	-3	Zone Total général

Note : Vous devez passer une valeur dans *numColonne* même lorsque vous passez -4 ou -5 dans le paramètre *numLigne* (dans ce cas la valeur de *numColonne* est inutilisée).

Note : Dans les tableaux croisés, le principe est similaire sauf pour les valeurs des lignes, qui sont toujours positives.

Passez dans *propriété* le numéro de la propriété de texte à lire. Vous pouvez utiliser les constantes du thème **QR Propriétés de texte**. Dans le tableau ci-dessous, la colonne Commentaire indique les valeurs pouvant être retournées :

Constante	Type	Valeur	Comment
_o_qr police	Entier long	1	Obsolète depuis 4D v14R3 (utiliser qr nom de police)
qr couleur de fond	Entier long	8	Numéro de couleur de fond
qr couleur de fond alternée	Entier long	9	Numéro de couleur de fond alternée
qr couleur de texte	Entier long	6	Numéro de couleur (Entier long)
qr gras	Entier long	3	Attribut gras (0 ou 1)
qr italique	Entier long	4	Attribut italique (0 ou 1)
qr justification	Entier long	7	Attribut de justification (0 = par défaut, 1 = gauche, 2 = centre et 3 = droite)
qr nom de police	Entier long	10	Nom de police tel que retourné par exemple par la commande **LISTE DES POLICES**.
qr souligné	Entier long	5	Attribut souligné (0 ou 1)
qr taille police	Entier long	2	Taille de police en points (9 à 255)

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numColonne* est incorrect, l'erreur -9852 est générée.
Si le paramètre *numLigne* est incorrect, l'erreur -9853 est générée.
Si le paramètre *propriété* est incorrect, l'erreur -9854 est générée.

QR Lire propriete zone

QR Lire propriete zone (zone ; propriété) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
propriété	Entier long	⇒	Elément d'interface
Résultat	Entier long	⇐	1 = affiché, 0 = caché

Description

La commande **QR Lire propriete zone** retourne 0 si l'élément d'interface (barre d'outils ou barre de menus) dont la référence est passée dans le paramètre *propriété* est masqué dans la *zone*, sinon elle retourne 1.

La barre de menus et les barres d'outils sont numérotées de 1 à 6 (de haut en bas) et la valeur 7 est associée au menu contextuel. Vous pouvez utiliser les constantes du thème **QR Propriétés de zone** afin de désigner l'élément d'interface :

Constante	Type	Valeur	Comment
qr barre menu	Entier long	1	Affichage de la barre de menus (Affichée=1, Cachée=0)
qr barre outils colonnes	Entier long	6	Affichage de la barre d'outils Colonnes (Affichée=1, Cachée=0)
qr barre outils couleurs	Entier long	5	Affichage de la barre d'outils Couleurs de fond (Affichée=1, Cachée=0)
qr barre outils opérateurs	Entier long	4	Affichage de la barre d'outils Opérateurs (Affichée=1, Cachée=0)
qr barre outils standard	Entier long	2	Affichage de la barre d'outils standard (Affichée=1, Cachée=0)
qr barre outils style	Entier long	3	Affichage de la barre d'outils Style (Affichée=1, Cachée=0)
qr menus contextuels	Entier long	7	Affichage des menus contextuels (Affichés=1, Cachés=0)

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *propriété* est incorrect, l'erreur -9852 est générée.

QR LIRE SELECTION

QR LIRE SELECTION (zone ; gauche ; haut {; droite {; bas}})

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
gauche	Entier long	⇐	Limite gauche
haut	Entier long	⇐	Limite supérieure
droite	Entier long	⇐	Limite droite
bas	Entier long	⇐	Limite inférieure

Description

La commande **QR LIRE SELECTION** retourne les coordonnées de la sélection courante de la *zone*.

gauche retourne le numéro de la colonne représentant la limite gauche de la sélection. Si *gauche* vaut 0, la ligne entière est sélectionnée.
haut retourne le numéro de la ligne représentant la limite supérieure de la sélection. Si *haut* vaut 0, la colonne entière est sélectionnée.

Note : Si les paramètres *gauche* and *haut* valent 0, la totalité de la zone est sélectionnée.

droite retourne le numéro de la colonne représentant la limite droite de la sélection.
bas retourne le numéro de la ligne représentant la limite inférieure de la sélection.

Note : Si la *zone* ne contient aucune sélection, les paramètres *gauche*, *haut*, *droite* et *bas* retournent -1.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR Lire statut commande

QR Lire statut commande (zone ; numCommande {; valeur}) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
numCommande	Entier long	⇒	Numéro de commande
valeur	Entier long, Texte	⇐	Valeur du sous-élément sélectionné
Résultat	Entier long	⇒	Statut de la commande

Description

La commande **QR Lire statut commande** retourne 0 si la commande désignée par le paramètre *numCommande* est inactivée et 1 si elle est activée.

valeur retourne la valeur du sous-élément sélectionné, le cas échéant. Par exemple, si la commande sélectionnée est la liste déroulante des polices (1000) et que la police choisie est l'Arial, *valeur* vaut “Arial” ; si la commande sélectionnée est le menu des couleurs (1002, 1003 ou 1004), *valeur* retourne le numéro de la couleur.

La commande **QR Lire statut commande** peut être utilisée dans deux types de contextes :

- comme simple instruction pour déterminer si une commande est active ou non.
- dans une méthode installée par **QR APPELER SUR COMMANDE** afin de connaître le sous-élément sélectionné. Dans cette méthode, *$1* contient la référence de la *zone* et *$2* le numéro de la commande.

Passez dans paramètre *numCommande* une des constantes du thème **QR Commandes** :

Constante	Type	Valeur
qr cmd ajouter colonne	Entier long	2608
qr cmd aligné à droite	Entier long	505
qr cmd aligné à gauche	Entier long	503
qr cmd aligné par défaut	Entier long	512
qr cmd apercu	Entier long	2007
qr cmd cacher colonne	Entier long	2602
qr cmd cacher ligne	Entier long	2607
qr cmd centré	Entier long	504
qr cmd déplacer à droite	Entier long	3003
qr cmd déplacer à gauche	Entier long	3002
qr cmd destination 4D View	Entier long	2503
qr cmd destination fichier	Entier long	2501
qr cmd destination fichier HTML	Entier long	2504
qr cmd destination graphe	Entier long	2502
qr cmd destination imprimante	Entier long	2500
qr cmd écart type	Entier long	513
qr cmd éditer colonne	Entier long	2603
qr cmd encadrements	Entier long	2609
qr cmd enregistrer	Entier long	2002
qr cmd enregistrer sous	Entier long	2003
qr cmd entete et pied de page	Entier long	2005
qr cmd espacement des totaux	Entier long	2610
qr cmd executer	Entier long	2008
qr cmd format	Entier long	2606
qr cmd gras	Entier long	500
qr cmd insérer colonne	Entier long	2600
qr cmd italique	Entier long	501
qr cmd largeur automatique	Entier long	2605
qr cmd liste déroulante police	Entier long	1000
qr cmd max	Entier long	509
qr cmd min	Entier long	508
qr cmd mise en forme	Entier long	2611
qr cmd mise en page	Entier long	2006
qr cmd moyenne	Entier long	507
qr cmd nombre	Entier long	510
qr cmd normal	Entier long	511
qr cmd nouveau	Entier long	2000
qr cmd objet couleur fond	Entier long	1003
qr cmd objet couleur fond alt	Entier long	1004
qr cmd objet couleur texte	Entier long	1002
qr cmd ouvrir	Entier long	2001
qr cmd palette colonnes	Entier long	2054
qr cmd palette couleurs de fond	Entier long	2052
qr cmd palette opérateurs	Entier long	2051
qr cmd palette standard	Entier long	2053
qr cmd palette style	Entier long	2050
qr cmd somme	Entier long	506
qr cmd souligné	Entier long	502
qr cmd supprimer colonne	Entier long	2601
qr cmd valeurs répétées	Entier long	2604
qr cmd version enregistrée	Entier long	2004

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numCommande* est incorrect, l'erreur -9852 est générée.

QR Lire table etat

QR Lire table etat (zone) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
Résultat	Entier long	⇐	Numéro de table

Description

La commande **QR Lire table etat** retourne le numéro de la table courante de l'état désigné par le paramètre *zone*.
Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR LIRE TRIS

QR LIRE TRIS (zone ; tabColonnes ; tabTris)

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
tabColonnes	Tableau réel	⇦	Colonnes triées
tabTris	Tableau réel	⇦	Ordres de tris

Description

La commande **QR LIRE TRIS** remplit deux tableaux réels :

- *tabColonnes*
  Ce tableau contient toutes les colonnes auxquelles un ordre de tri a été associé.
- *tabTris*
  Chaque élément de ce tableau fournit l'ordre de tri courant de la colonne correspondante.
  - si *tabTris{$i}* vaut 1, le tri est croissant.
  - si *tabTris{$i}* vaut -1, le tri est décroissant.

Etats tableaux croisés

Avec ce type d'état, les tableaux ne peuvent pas comporter plus de deux éléments puisque les tris ne peuvent être effectués que sur les colonnes (1) et les lignes (2) (valeurs pour *tabColonnes*).

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR Lire type etat

QR Lire type etat (zone) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
Résultat	Entier long	⇐	Type d'état

Description

La commande **QR Lire type etat** retourne le *type* d'état présent dans la *zone*.

Vous pouvez comparer le résultat avec les constantes du thème **QR Types d'états** :

Constante	Type	Valeur
qr état en liste	Entier long	1
qr état tableau croisé	Entier long	2

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

QR Nombre de colonnes

QR Nombre de colonnes (zone) -> Résultat

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone
Résultat	Entier long	⇦	Nombre de colonnes dans la zone

Description

La commande **QR Nombre de colonnes** retourne le nombre de colonnes présentes dans l'état rapide désigné par le paramètre *zone*.
Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

Exemple

L'exemple suivant permet d'ajouter une colonne supplémentaire à droite de la dernière colonne de la zone :

```
$NbCol:=QR Nombre de colonnes(MaZone)
QR INSERER COLONNE(MaZone;$NbCol+1;->[Table 1]Noms)
```

QR SUPPRIMER COLONNE

QR SUPPRIMER COLONNE (zone ; numColonne)

Paramètre	Type		Description
zone	Entier long	⇒	Référence de la zone
numColonne	Entier long	⇒	Numéro de colonne

Description

La commande **QR SUPPRIMER COLONNE** supprime de la *zone* la colonne dont le numéro a été passé dans *numColonne*. Cette commande ne peut pas être utilisée avec les états en tableau croisé.

Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.
Si le paramètre *numColonne* est incorrect, l'erreur -9852 est générée.

Exemple

Cet exemple supprime la troisième colonne de l'état :

```
Si(QR Lire type etat(MaZone)=gr état en liste)
   QR SUPPRIMER COLONNE(MaZone;3)
Fin de si
```

QR SUPPRIMER ZONE HORS ECRAN

QR SUPPRIMER ZONE HORS ECRAN (zone)

Paramètre	Type		Description
zone	Entier long	⇨	Référence de la zone

Description

La commande **QR SUPPRIMER ZONE HORS ECRAN** efface de la mémoire la zone hors écran dont la référence a été passée dans le paramètre *zone*.
Si un numéro de *zone* invalide est passé, l'erreur -9850 est générée.

Evénements formulaire

- Activation
- Appel exterieur
- APPELER CONTENEUR SOUS FORMULAIRE
- Apres
- Attente relachement clic Nouveauté 16.0
- Avant
- Clic contextuel
- Clic droit
- Desactivation
- En entete
- En pied
- En rupture
- Evenement formulaire Modifié 16.0
- FIXER MINUTEUR
- Nombre de clics
- *_o_Pendant*

Activation

Activation -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si le cycle d'exécution est en activation

Description

Activation retourne Vrai dans une méthode formulaire lorsque la fenêtre contenant le formulaire passe au premier plan.

Note : Cette commande équivaut à utiliser la fonction **Evenement formulaire** et tester si elle retourne l'événement Sur activation.

ATTENTION : N'appelez pas de commandes telles que **TRACE** ou **ALERTE** dans la phase **Activation** d'un formulaire, car cela provoquerait une boucle sans fin.

Note : Si vous voulez que le cycle d'exécution **Activation** soit généré, assurez-vous que la propriété d'événement Sur activation du formulaire et/ou des objet(s) est sélectionnée en mode Développement.

Appel exterieur

Appel exterieur -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si le cycle d'exécution est appel extérieur

Description

Appel exterieur retourne Vrai pour le cycle d'exécution Appel extérieur.

Si vous voulez que le cycle d'exécution **Appel exterieur** soit généré, vérifiez que la propriété d'événement Sur appel extérieur du formulaire et/ou des objets est sélectionnée en mode Développement.

Note : Cette fonction équivaut à utiliser la fonction **Evenement formulaire** et tester si elle retourne un événement tel que Sur appel extérieur.

APPELER CONTENEUR SOUS FORMULAIRE

APPELER CONTENEUR SOUS FORMULAIRE (événement)

Paramètre	Type		Description
événement	Entier long	⇨	Evénement à transmettre

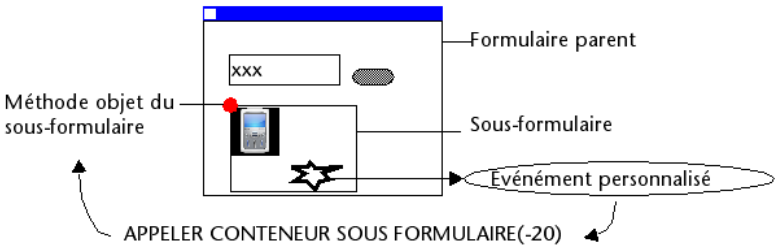
Description

La commande **APPELER CONTENEUR SOUS FORMULAIRE** permet à une instance de sous-formulaire d'envoyer l'*événement* à l'objet sous-formulaire qui la contient. L'objet sous-formulaire peut alors traiter l'*événement* dans le contexte du formulaire parent.

Cette commande doit être placée dans la méthode formulaire du sous-formulaire ou dans la méthode objet d'un des objets du sous-formulaire. L'événement sera reçu uniquement dans la méthode objet du conteneur du sous-formulaire.

Vous pouvez passer dans *événement* tout événement formulaire prédéfini de 4D (vous pouvez utiliser les constantes du thème **Evénements formulaire**) ou toute valeur correspondant à un événement personnalisé. Dans le premier cas, l'événement doit être coché pour le sous-formulaire. Dans le cas d'un événement personnalisé, il est conseillé de passer une valeur négative dans *événement* afin de ne pas risquer d'interférer avec des numéros d'événements existants ou à venir de 4D.

Principe d'exécution de la commande **APPELER CONTENEUR SOUS FORMULAIRE** *:*



Apres

Apres -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Retourne Vrai si le cycle d'exécution est après

Description

Apres retourne Vrai pour le cycle d'exécution Après.

Si vous souhaitez que la phase **Apres** du cycle d'exécution soit générée, assurez-vous que l'événement Sur validation a bien été sélectionné, en mode Développement, dans les propriétés du formulaire et/ou des objets concernés.

Note : Cette commande équivaut à utiliser la fonction **Evenement formulaire** et tester si elle retourne l'événement Sur validation.

Attente relachement clic

Attente relachement clic -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si l'objet est en attente d'un relâchement souris, Faux sinon

Description

La commande **Attente relachement clic** retourne **Vrai** si l'objet courant a été cliqué et que le bouton de la souris n'a pas été relâché, alors que la fenêtre contenant l'objet a le focus. Sinon la commande retourne **Faux**, en particulier si la fenêtre contenant l'objet a perdu le focus avant que le bouton soit relâché.

Cette commande doit être appelée dans le contexte de l'objet courant. Elle est destinée à être utilisée conjointement avec l'événement formulaire Sur relâchement bouton, disponible pour les champs et variables Image. Elle permet de gérer par programmation les cas où l'utilisateur clique et commence à effectuer un glisser dans une zone image (champ ou variable), et que cette action est interrompue par un événement extérieur, comme par exemple l'affichage d'une boîte de dialogue d'alerte. Dans ce cas, l'état interne de l'objet peut être suspendu indéfiniment car il attend l'événement Sur relâchement bouton qui n'arrive jamais. Pour éviter ce problème, vous devez protéger votre code de déplacement de souris à l'aide de la commande **Attente relachement clic** qui vous assure de son exécution dans un contexte valide.

Exemple

Le code suivant peut être utilisé pour gérer le suivi de la souris dans un objet image :

```
 //Méthode objet de l'objet image
C_ENTIER LONG(vLtracking) //drapeau du mode Suivi
Au cas ou
   :(Evenement formulaire=Sur clic)
      Si(Attente relachement clic) // le bouton de la souris n'a pas été relâché
         vLtracking:=1 //nous sommes en mode Suivi
 //... Ecrivez ici le code d'initialisation pour la fonction de Suivi
      Fin de si
   :(Evenement formulaire=Sur survol)
      Si(vLtracking=1) //nous sommes en mode Suivi
         Si(Non(Attente relachement clic)) // on n'aura jamais le relâchement de la souris
            vLtracking:=0 // on stoppe le mode Suivi
 //... Ecrivez ici le code pour le traitement ou l'annulation de l'action de suivi
         Sinon //l'objet est toujours en attente du relâchement de la souris
 //... Ecrivez ici le code pour le suivi
         Fin de si
      Fin de si
   :(Evenement formulaire=Sur relâchement bouton) // le bouton de la souris a été relâché
 //... Ecrivez ici le code pour le suivi du relâchement souris
      vLtracking:=0 //Fin du suivi
Fin de cas
```

Avant

Avant -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si le cycle d'exécution est avant

Description

Avant retourne Vrai pour le cycle d'exécution Avant.

Si vous souhaitez que la phase **Avant** du cycle d'exécution soit générée, assurez-vous que l'événement Sur chargement a bien été sélectionné, en mode Développement, dans les propriétés du formulaire et/ou des objets concernés.

Note : Cette commande équivaut à utiliser la fonction **Evenement formulaire** et tester si elle retourne l'événement Sur chargement.

Clic contextuel

Clic contextuel -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si un clic contextuel a été détecté, sinon Faux

Description

La commande **Clic contextuel** retourne Vrai si un clic de type contextuel a été effectué :

- Sous Windows et Mac OS, les clics contextuels sont effectués avec le bouton droit de la souris.
- Sous Mac OS, des clics contexuels peuvent également être générés à l'aide de la combinaison **Control+clic**.

Cette commande doit être appelée uniquement dans le cadre de l'événement formulaire Sur clic. Il est donc nécessaire de vérifier en mode Développement que cet événement a bien été coché dans les Propriétés du formulaire et/ou de l'objet.

Exemple

Cette méthode, associée à une zone de défilement, permet de changer la valeur d'un élément de tableau à l'aide d'un menu contextuel :

```
Si(Clic contextuel)
   Si(Pop up menu("Vrai;Faux")=1)
      monTableau{monTableau}:="Vrai"
   Sinon
      monTableau{monTableau}:="Faux"
   Fin de si
Fin de si
```

Clic droit

Clic droit -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si un clic droit a été détecté, sinon Faux

Description

La commande **Clic droit** retourne Vrai si un clic effectué avec le bouton droit de la souris a été effectué.

Cette commande doit être appelée uniquement dans le cadre de l'événement formulaire Sur clic. Il est donc nécessaire de vérifier en mode Développement que cet événement a bien été coché dans les Propriétés du formulaire et/ou de l'objet.

Desactivation

Desactivation -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si le cycle d'exécution est en désactivation

Description

Desactivation retourne Vrai dans une méthode formulaire ou méthode objet lorsque la fenêtre appartenant au process du premier plan, contenant le formulaire, passe à l'arrière-plan.

Si vous voulez que le cycle d'exécution **Desactivation** soit généré, vérifiez que la propriété d'événement Sur désactivation du formulaire et/ou des objets est sélectionnée en mode Développement.

Note : Cette commande équivaut à utiliser la fonction **Evenement formulaire** et tester si elle retourne l'événement Sur désactivation.

En entete

En entete -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si le cycle d'exécution est en entête

Description

En entete retourne Vrai pour le cycle d'exécution En entête.

Si vous souhaitez que le cycle d'exécution **En entete** soit généré, assurez-vous que l'événement formulaire Sur entête a bien été activé dans la boîte de dialogue des propriétés du formulaire ou du (des) objet(s) sélectionné(s), en mode Développement.

Note : Cette commande équivaut à utiliser la commande **Evenement formulaire** et tester si elle retourne l'événement Sur entête.

En pied

En pied -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Vrai si le cycle d'exécution est en pied

Description

En pied retourne Vrai pour le cycle d'exécution En pied.

Si vous voulez que le cycle d'exécution **En pied** soit généré, vérifiez que la propriété d'événement Sur impression pied de page du formulaire et/ou des objets est sélectionnée en mode Développement.

Note : Cette fonction équivaut à utiliser la fonction **Evenement formulaire** et tester si elle retourne l'événement Sur impression pied de page.

En rupture

En rupture -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si le cycle d'exécution est en rupture

Description

En rupture retourne Vrai pour le cycle d'exécution En rupture.

Si vous souhaitez que le cycle d'exécution **En rupture** soit généré, assurez-vous que l'événement formulaire Sur impression sous total a bien été activé dans la boîte de dialogue des propriétés du formulaire ou du (des) objet(s) sélectionné(s), en mode Développement.

Note : Cette commande équivaut à utiliser la commande **Evenement formulaire** et tester si elle retourne l'événement Sur impression sous total.

Evenement formulaire

Evenement formulaire -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Numéro d'événement formulaire

Description

Evenement formulaire retourne une valeur numérique qui identifie le type d'événement formulaire qui vient de se produire. Généralement, **Evenement formulaire** s'utilise dans une méthode formulaire ou une méthode objet.

4D fournit des constantes prédéfinies (placées dans le thème **Evénements formulaire**) permettant de comparer les valeurs retournées par la commande **Evenement formulaire**. Certains événements sont génériques (générés pour tout type d'objet), d'autres sont spécifiques à un type d'objet particulier.

Constante	Type	Valeur	Comment
Sur chargement	Entier long	1	Le formulaire s'affiche ou s'imprime
Sur relâchement bouton	Entier long	2	(*Images uniquement*) L'utilisateur vient de relâcher le bouton de la souris dans un objet Image
Sur validation	Entier long	3	La saisie des données dans l'enregistrement est validée
Sur clic	Entier long	4	Un clic est survenu sur un objet
Sur entête	Entier long	5	L'en-tête du formulaire va être imprimé ou affiché
Sur impression sous total	Entier long	6	Une rupture du formulaire va être imprimée
Sur impression pied de page	Entier long	7	Le pied de page du formulaire va être imprimé
Sur affichage corps	Entier long	8	Un enregistrement va être affiché dans la liste ou une ligne va être affichée dans la list box
Sur appel extérieur	Entier long	10	Le formulaire a reçu un appel de la commande **APPELER PROCESS**
Sur activation	Entier long	11	La fenêtre du formulaire passe au premier plan
Sur désactivation	Entier long	12	La fenêtre du formulaire passe en arrière-plan
Sur double clic	Entier long	13	Un double-clic est survenu sur un objet
Sur perte focus	Entier long	14	Un objet de formulaire perd le focus
Sur gain focus	Entier long	15	Un objet de formulaire prend le focus
Sur déposer	Entier long	16	Des données sont déposées sur un objet
Sur avant frappe clavier	Entier long	17	Un caractère vient d'être saisi dans l'objet qui a le focus. **Lire texte edite** retourne le contenu sans ce caractère
Sur menu sélectionné	Entier long	18	Une commande de menu a été sélectionnée
Sur appel zone du plug in	Entier long	19	Un plug-in demande que sa méthode objet soit exécutée
Sur données modifiées	Entier long	20	Les données d'un objet ont été modifiées
Sur glisser	Entier long	21	Des données sont glissées sur un objet
Sur case de fermeture	Entier long	22	On a cliqué sur la case de fermeture de la fenêtre
Sur impression corps	Entier long	23	Le corps du formulaire va être imprimé
Sur libération	Entier long	24	Le formulaire se referme et est déchargé
Sur ouverture corps	Entier long	25	Le formulaire détaillé associé au formulaire sortie ou à la listbox est sur le point d'être ouvert
Sur fermeture corps	Entier long	26	Le formulaire détaillé se referme et on retourne au formulaire sortie
Sur minuteur	Entier long	27	Le nombre de ticks défini par **FIXER MINUTEUR** est atteint
Sur après frappe clavier	Entier long	28	Un caractère vient d'être saisi dans l'objet qui a le focus. **Lire texte edite** retourne le contenu avec ce caractère.
Sur redimensionnement	Entier long	29	La fenêtre du formulaire est redimensionnée
Sur après tri	Entier long	30	(*List box uniquement*) Un tri standard vient d'être effectué dans une colonne de list box

Constante	Type	Valeur	Comment
			• *List box* : la sélection courante de lignes ou de colonnes est modifiée
Sur nouvelle sélection	Entier long	31	• *Enregistrements en liste* : l'enregistrement courant ou la sélection courante de lignes est modifié(e) dans un formulaire en liste ou un sous-formulaire • *Liste hiérarchique* : la sélection dans la liste est modifiée à la suite d'un clic ou de la frappe d'une touche au clavier • *Variable ou champ saisissable* : la sélection de texte ou la position du curseur dans la zone est modifiée à la suite d'un clic ou de la frappe d'une touche au clavier
Sur déplacement colonne	Entier long	32	(*List box uniquement*) Une colonne de list box est déplacée par l'utilisateur via le glisser-déposer
Sur redimensionnement colonne	Entier long	33	(*List box uniquement*) La largeur d'une colonne de list box est modifiée par l'utilisateur via la souris
Sur déplacement ligne	Entier long	34	(*List box uniquement*) Une ligne de list box est déplacée par l'utilisateur via le glisser-déposer
Sur début survol	Entier long	35	Le curseur de la souris entre dans la zone graphique d'un objet
Sur fin survol	Entier long	36	Le curseur de la souris sort de la zone graphique d'un objet
Sur survol	Entier long	37	Le curseur de la souris a bougé d'au moins un pixel OU une touche de modification (Ctrl, Alt, Verr Maj.) a été enfoncée. Si l'événement est coché pour un objet uniquement, il n'est généré que lorsque le curseur se trouve dans la zone graphique de l'objet
Sur clic alternatif	Entier long	38	• *Boutons 3D* : La zone "flèche" d'un bouton 3D reçoit un clic • *List box* : Dans une colonne tableau d'objets, un bouton d'ellipse (attribut "alternateButton") reçoit un clic **Note**: Les boutons d'ellipses sont disponibles à partir de la v15 uniquement.
Sur clic long	Entier long	39	(*Boutons 3D uniquement*) Un bouton 3D reçoit un clic et le bouton de la souris reste enfoncé pendant un certain laps de temps
Sur chargement ligne	Entier long	40	En mode saisie en liste, un enregistrement est chargé en modification (l'utilisateur a cliqué sur la ligne de l'enregistrement et un champ passe en édition)
Sur avant saisie	Entier long	41	(*List box uniquement*) Une cellule de list box est sur le point de passer en mode édition
Sur clic entête	Entier long	42	(*List box uniquement*) Un clic est survenu dans l'en-tête d'une colonne de list box
Sur déployer	Entier long	43	(*Listes hiérarchiques et List box hiérarchiques*) Un élément de liste hiérarchique ou de list box hiérarchique a été déployé via un clic ou une touche du clavier
Sur contracter	Entier long	44	(*Listes hiérarchiques et list box hiérarchiques*) Un élément de liste hiérarchique ou de list box hiérarchique a été contracté via un clic ou une touche du clavier
Sur après modification	Entier long	45	Le contenu de l'objet saisissable qui a le focus vient d'être modifié
Sur début glisser	Entier long	46	Un objet est en cours de glisser
Sur début chargement URL	Entier long	47	(*Zones Web uniquement*) Un nouvel URL est chargé dans la zone Web
Sur chargement ressource URL	Entier long	48	(*Zones Web uniquement*) Une nouvelle ressource est chargée dans la zone Web
Sur fin chargement URL	Entier long	49	(*Zones Web uniquement*) Toutes les ressources de l'URL ont été chargées
Sur erreur chargement URL	Entier long	50	(*Zones Web uniquement*) Une erreur s'est produite durant le chargement de l'URL
Sur filtrage URL	Entier long	51	(*Zones Web uniquement*) Un URL a été bloqué par la zone Web
Sur ouverture lien externe	Entier long	52	(*Zones Web uniquement*) Un URL externe a été ouvert dans le navigateur
Sur refus ouverture fenêtre	Entier long	53	(*Zones Web uniquement*) Une fenêtre pop up a été bloquée
Sur modif variable liée	Entier long	54	La variable liée à un sous-formulaire est modifiée.
_o_Sur bouton barre outils Mac	Entier long	55	*** Constante obsolète ***
Sur changement page	Entier long	56	On a changé de page courante dans le formulaire
Sur clic pied	Entier long	57	(*List box uniquement*) Un clic est survenu dans le pied d'une list box ou d'une colonne de list box
Sur action suppression	Entier long	58	(*Listes hiérarchiques et List box*) L'utilisateur a demandé à supprimer un élément
Sur défilement	Entier long	59	*Variables ou champs image et List Box* : L'utilisateur fait défiler le contenu du champ image, de la variable image ou de la list box à l'aide de la souris ou d'une touche du clavier.

Note : Les événements spécifiques des formulaires de sortie ne peuvent pas être utilisés avec les **formulaires projet**. Il s'agit de : Sur affichage corps, Sur ouverture corps, Sur fermeture corps, Sur chargement ligne, Sur entête, Sur impression corps, Sur impression sous total, Sur impression pied de page.

Evénements et méthodes

Lorsqu'un événement formulaire se produit, 4D effectue les actions suivantes :

- En premier lieu, il examine chaque objet du formulaire et appelle la méthode de ceux dont la propriété d'événement correspondante a été sélectionnée

et qui sont impliqués dans l'événement.
- Ensuite, il appelle la méthode formulaire si la propriété d'événement correspondante a été sélectionnée pour le formulaire.

Les différentes méthodes objet ne sont pas appelées dans un ordre particulier. La règle est que les méthodes objet sont toujours appelées avant la méthode formulaire. Dans le cas des sous-formulaires, les méthodes objet du formulaire sortie du sous-formulaire sont d'abord appelées, puis la méthode formulaire du formulaire sortie, puis enfin 4D appelle les méthodes objet du formulaire parent. Autrement dit, lorsqu'un objet est un sous-formulaire, 4D utilise la même règle pour les méthodes formulaire et objet dans le sous-formulaire.

Lorsque, pour un événement particulier, la propriété d'événement du formulaire n'est pas sélectionnée, cela n'empêche pas les appels aux méthodes des objets pour lesquels l'événement est sélectionné. Autrement dit, la sélection ou la désélection d'un événement au niveau du formulaire n'a pas d'effet sur les propriétés d'événements des objets.

ATTENTION : Ce principe ne s'applique pas aux événements Sur chargement et Sur libération. Ces événements ne seront générés pour un objet que si les propriétés d'événement correspondantes ont été sélectionnées à la fois pour l'objet et pour le formulaire auquel il appartient. Si les propriétés sont sélectionnées pour l'objet uniquement, les événements ne seront pas générés ; ces deux événements doivent être sélectionnés au niveau du formulaire.

Le nombre d'objets impliqués par un événement dépend de la nature de l'événement. En particulier :

- Pour l'événement Sur chargement, les méthodes objet de tous les objets du formulaire (sur toutes les pages) pour lequels la propriété d'événement Sur chargement est sélectionnée seront appelées. Si l'événement Sur chargement est sélectionné pour le formulaire, la méthode formulaire sera appelée.
- Pour les événements Sur activation ou Sur redimensionnement, aucune méthode objet ne sera appelée car ces événements s'appliquent au formulaire, pas à un objet en particulier. Par conséquent, si ces événements sont sélectionnés pour le formulaire, seule la méthode formulaire sera appelée.
- L'événement Sur minuteur n'est généré que si la méthode formulaire contient un appel préalable à la commande **FIXER MINUTEUR**. Seule la méthode formulaire reçoit cet événement, aucune méthode objet ne sera appelée.
- Pour l'événement Sur glisser, seule la méthode de l'objet déposable impliqué par l'événement sera appelée (si la propriété d'événement "Déposable" est sélectionnée pour l'objet). La méthode formulaire ne sera pas appelée.
- A l'inverse, pour l'événement Sur début glisser, la méthode objet ou la méthode formulaire de l'objet glissé sera appelée (si la propriété d'événement "Glissable" est sélectionnée pour l'objet).

ATTENTION : Contrairement aux autres événements, pendant un événement Sur début glisser, la méthode appelée est exécutée dans le contexte du process de l'objet source du glisser-déposer, et non dans celui du process de l'objet de destination. Pour plus d'informations, reportez-vous à la section **Présentation du Glisser-Déposer**.

- Si les événements Sur début survol, Sur survol et Sur fin survol sont cochés pour le formulaire, ils sont générés pour chaque objet du formulaire. S'ils sont cochés pour un objet, ils sont générés pour cet objet uniquement. En cas de superposition d'objets, l'événement est généré par le premier objet capable de le gérer dans l'ordre des plans du haut vers le bas. Les objets rendus invisibles par la commande **OBJET FIXER VISIBLE** ne génèrent pas ces événements. Pendant la saisie d'un objet, les autres objets peuvent recevoir les événements de survol en fonction de la position de la souris. A noter que l'événement Sur survol est généré lorsque le curseur de la souris est déplacé mais également lorsque l'utilisateur appuie sur une touche de modification telle que **Maj**, **Verr. Maj**, **Ctrl** ou **Option** (ce principe permet notamment de gérer les glisser-déposer de type copie ou déplacement).
- Enregistrements en liste : l'enchaînement d'appels des méthodes et des événements formulaires dans les formulaires liste affichés via **MODIFIER SELECTION** / **VISUALISER SELECTION** et les sous-formulaires est le suivant :

  Pour chaque objet de la zone d'en-tête :
  - Méthode objet avec événement Sur entête
  - Méthode formulaire avec événement Sur entête

  Pour chaque enregistrement :
  - Pour chaque objet de la zone de corps :
    - Méthode objet avec événement Sur affichage corps
  - Méthode formulaire avec événement Sur affichage corps

- L'appel depuis les événements Sur affichage corps et Sur entête d'une commande 4D provoquant l'affichage d'une boîte de dialogue est interdit et provoque une erreur de syntaxe. Les commandes concernées sont notamment : **ALERTE**, **DIALOGUE**, **CONFIRMER**, **Demander**, **AJOUTER ENREGISTREMENT**, **MODIFIER ENREGISTREMENT**, **VISUALISER SELECTION** et **MODIFIER SELECTION**.
- Sur changement page : cet événement est disponible au niveau des formulaires uniquement (il est appelé dans la méthode formulaire). Il est généré à chaque changement de page courante du formulaire (à la suite d'un appel à la commande **FORM ALLER A PAGE** ou d'une action standard de navigation). A noter que l'événement est généré après le chargement complet de la page, c'est-à-dire une fois que tous les objets qu'elle contient sont initialisés (y compris les zones Web). Cet événement est utile pour exécuter du code qui nécessite que tous les objets soient initialisés au préalable. Il permet également d'optimiser l'application en n'exécutant du code (par exemple une recherche) qu'après l'affichage d'une page spécifique du formulaire et non dès le chargement de la page 1. Si l'utilisateur n'accède pas à la page, le code n'est pas exécuté.

Le tableau suivant résume, pour chaque type d'événement, l'appel des méthodes formulaire et objet :

Evénement	Méthode(s) objet	Méthode formulaire	Quel(s) objet(s)
Sur chargement	Oui	Oui	Tous
Sur libération	Oui	Oui	Tous
Sur validation	Oui	Oui	Tous
Sur clic	Oui (si cliquable ou saisissable) (*)	Oui	Seul l'objet impliqué
Sur double clic	Oui (si cliquable ou saisissable) (*)	Oui	Seul l'objet impliqué
Sur avant frappe clavier	Oui (si saisissable) (*)	Oui	Seul l'objet impliqué
Sur après frappe clavier	Oui (si saisissable) (*)	Oui	Seul l'objet impliqué
Sur après modification	Oui (si saisissable) (*)	Oui	Seul l'objet impliqué
Sur gain focus	Oui (si tabulable) (*)	Oui	Seul l'objet impliqué
Sur perte focus	Oui (si tabulable) (*)	Oui	Seul l'objet impliqué
Sur activation	Jamais	Oui	Aucun
Sur désactivation	Jamais	Oui	Aucun
Sur appel extérieur	Jamais	Oui	Aucun
Sur changement page	Jamais	Oui	Aucun
Sur début glisser	Oui (si glissable) (**)	Oui	Seul l'objet impliqué
Sur déposer	Oui (si déposable) (**)	Oui	Seul l'objet impliqué
Sur glisser	Oui (si déposable) (**)	Jamais	Seul l'objet impliqué
Sur début survol	Oui	Oui	Tous
Sur survol	Oui	Oui	Tous
Sur fin survol	Oui	Oui	Tous
Sur relâchement bouton	Oui	Jamais	Seul l'objet impliqué
Sur menu sélectionné	Jamais	Oui	Aucun
Sur modif variable liée	Jamais	Oui	Aucun
Sur données modifiées	Oui (si modifiable) (*)	Oui	Seul l'objet impliqué
Sur appel zone du plug in	Oui	Oui	Seul l'objet impliqué
Sur entête	Oui	Oui	Tous
Sur impression corps	Oui	Oui	Tous
Sur impression sous total	Oui	Oui	Tous
Sur impression pied de page	Oui	Oui	Tous
Sur case de fermeture	Jamais	Oui	Aucun
Sur affichage corps	Oui	Oui	Tous
Sur ouverture corps	Non sauf List box	Oui	Aucun sauf List box
Sur fermeture corps	Non sauf List box	Oui	Aucun sauf List box
Sur redimensionnement	Jamais	Oui	Aucun
Sur nouvelle sélection	Oui (***)	Oui	Seul l'objet impliqué
Sur chargement ligne	Jamais	Oui	Aucun
Sur minuteur	Jamais	Oui	Aucun
Sur défilement	Oui	Jamais	Seul l'objet impliqué
Sur avant saisie	Oui (List box)	Jamais	Seul l'objet impliqué
Sur déplacement colonne	Oui (List box)	Jamais	Seul l'objet impliqué
Sur déplacement ligne	Oui (List box)	Jamais	Seul l'objet impliqué
Sur redimensionnement colonne	Oui (List box)	Jamais	Seul l'objet impliqué
Sur clic entête	Oui (List box)	Jamais	Seul l'objet impliqué
Sur clic pied	Oui (List box)	Jamais	Seul l'objet impliqué
Sur après tri	Oui (List box)	Jamais	Seul l'objet impliqué
Sur clic long	Oui (Bouton 3D)	Oui	Seul l'objet impliqué
Sur clic alternatif	Oui (Bouton 3D et List box)	Jamais	Seul l'objet impliqué
Sur déployer	Oui (Liste hiér. et List box)	Jamais	Seul l'objet impliqué
Sur contracter	Oui (Liste hiér. et List box)	Jamais	Seul l'objet impliqué
Sur action suppression	Oui (Liste hiér. et List box)	Jamais	Seul l'objet impliqué
Sur chargement ressource URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur début chargement URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur erreur chargement URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur filtrage URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur fin chargement URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur ouverture lien externe	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur refus ouverture fenêtre	Oui (Zone Web)	Jamais	Seul l'objet impliqué

(*) Référez-vous ci-dessous au paragraphe "Evénements, objets et propriétés" pour plus d'informations.
(**) Référez-vous à la section **Présentation du Glisser-Déposer** pour plus d'informations.
(***) Seuls les objets de type List box, Liste hiérarchique et Sous-formulaire prennent en charge cet événement.

IMPORTANT : Gardez constamment à l'esprit que, pour chaque événement, la méthode d'un formulaire ou d'un objet est appelée si l'événement correspondant a été sélectionné pour le formulaire ou l'objet. L'avantage de désactiver des événements en mode Développement (en utilisant la Liste des propriétés de l'éditeur de formulaires) est que vous pouvez réduire de manière très importante le nombre d'appels aux méthodes et donc optimiser la vitesse d'exécution des formulaires.

Evénements, objets et propriétés

Une méthode objet est appelée si l'événement peut réellement se produire pour l'objet en fonction de sa nature et de ses propriétés. Ce paragraphe détaille les événements à utiliser pour gérer les différents types d'objets.

A noter que la Liste des propriétés de l'éditeur de formulaires n'affiche que les événements compatibles avec l'objet sélectionné ou le formulaire.

Objets cliquables

Les objets cliquables sont gérés principalement avec la souris. Ces objets sont les suivants :

- Variables ou champs saisissables de type Booléen
- Boutons, boutons par défaut, boutons radio, cases à cocher, grilles de boutons
- Boutons 3D, boutons radio 3D, cases à cocher 3D
- Pop up menus, pop up menus hiérarchiques, menus Images
- Listes déroulantes, menus,
- Zones de défilement, listes hiérarchiques, list box et colonnes de list box
- Boutons invisibles, boutons inversés, boutons radio image
- Thermomètres, règles, cadrans
- Onglets,
- Séparateurs.

Lorsque l'événement Sur clic ou Sur double clic est sélectionné pour un de ces objets, vous pouvez détecter et gérer les clics sur l'objet à l'aide de la commande **Evenement formulaire** qui retourne Sur clic ou Sur double clic selon le cas.
Si les deux événements sont sélectionnés pour un même objet, les événements Sur clic puis Sur double clic seront générés en cas de double-clic sur l'objet.

Note : A compter de 4D v14, les champs et variables saisissables contenant du texte (type texte, date, heure, numérique) génèrent également les événements Sur clic et Sur double clic.

Pour tous les objets cliquables, l'événement Sur clic se produit une fois que le bouton de la souris est relâché. Il y a cependant des exceptions :

- Avec les boutons invisibles et les onglets, l'événement Sur clic se produit dès qu'un clic a été détecté — sans attendre le relâchement du bouton de la souris.
- Avec les thermomètres, règles et cadrans, si le format d'affichage indique que la méthode objet doit être appelée pendant que vous faites glisser les curseurs de contrôle, l'événement Sur clic survient dès que le clic est détecté.

Dans le contexte de l'événement Sur clic, vous pouvez tester le nombre de clics effectués par l'utilisateur à l'aide de la commande **Nombre de clics**.

Note : Quelques objets peuvent être activés par le clavier. Une case à cocher, par exemple, une fois qu'elle a le focus, peut être sélectionnée à l'aide de la barre d'espace. Dans ce cas, l'événement Sur clic est quand même généré.

ATTENTION : Les combo-boxes ne sont pas considérées comme des objets cliquables. Une combo-box doit être perçue comme une zone de texte saisissable dont la liste déroulante fournit les valeurs par défaut. Par conséquent, vous gérez la saisie des données dans une combo-box à l'aide des événements Sur avant frappe clavier, Sur après frappe clavier et Sur données modifiées.

Note : A compter de 4D v13, les objets de type pop up/liste déroulante et menu déroulant hiérarchique peuvent générer l'événement Sur données modifiées. Ce principe vous permet de détecter l'activation de l'objet avec sélection d'une valeur différente de la valeur courante.

Objets saisissables par clavier

Les objets saisissables par clavier sont des objets dans lesquels vous saisissez des données par le clavier et pour lesquels vous pouvez filtrer les données au plus bas niveau en détectant les événements Sur après modification, Sur avant frappe clavier, Sur après frappe clavier et Sur nouvelle sélection.
Les objets et types de données saisissables sont les suivants :

- Tous les champs saisissables de type alpha, texte, date, heure, numérique ou (Sur après modification uniquement) image
- Toutes les variables saisissables de type alpha, texte, date, heure, numérique ou (Sur après modification uniquement) image
- Combo-boxes (sauf Sur nouvelle sélection)
- List boxes

Note : A compter de 4D v14, les champs et variables saisissables contenant du texte (type texte, date, heure, numérique) génèrent également les événements Sur clic et Sur double clic.

Note : Bien qu'objets "saisissables", les listes hiérarchiques ne gèrent pas les événements formulaire Sur après modification, Sur avant frappe clavier et Sur après frappe clavier (voir aussi le paragraphe "Listes hiérarchiques" ci-dessous).

- Sur avant frappe clavier et Sur après frappe clavier

Note : L'événement Sur après frappe clavier peut généralement être avantageusement remplacé par l'événement Sur après modification (cf. ci-dessous).
Une fois que les événements Sur avant frappe clavier et Sur après frappe clavier ont été sélectionnés pour un objet, vous pouvez détecter et gérer la saisie par le clavier dans l'objet à l'aide de la commande **Evenement formulaire** qui va retourner Sur avant frappe clavier puis Sur après frappe clavier (pour plus d'informations, reportez-vous à la description de la commande **Lire texte edite**). Ces événements sont également activés par les commandes du langage simulant une action utilisateur, telles que **GENERER FRAPPE CLAVIER**.
A noter que les modifications des utilisateurs non effectuées via le clavier (coller, glisser-déposer, etc.) ne sont pas prises en compte. Pour traiter ces événements, vous devez utiliser Sur après modification.

Note : Les événements Sur avant frappe clavier et Sur après frappe clavier ne sont pas générés lors de l'utilisation d'une méthode d'entrée. Une méthode d'entrée (ou IME, Input Method Editor) est un programme ou un composant système permettant la saisie de caractères complexes ou de symboles (par exemple japonais ou chinois) à l'aide d'un clavier occidental.

- Sur après modification
  Lorsqu'il est utilisé, cet événement est généré après chaque modification du contenu d'un objet saisissable, quelle que soit l'action à l'origine de cette modification, c'est-à-dire :
  - les actions d'édition standard entraînant une modification du contenu telles que coller, couper, effacer ou annuler ;
  - le déposer d'une valeur (action similaire au coller) ;
  - toute saisie au clavier effectuée par l'utilisateur ; dans ce cas, l'événement Sur après modification est généré après les événements Sur avant frappe clavier et Sur après frappe clavier s'ils sont utilisés.
  - une modification effectuée via une commande du langage simulant une action utilisateur (i.e. **GENERER FRAPPE CLAVIER**).
  Attention, les actions suivantes ne déclenchent PAS cet événement :
  - les actions d'édition ne modifiant pas le contenu de la zone, comme copier ou tout sélectionner, ou le glisser d'une valeur (action similaire au copier) ; en revanche, ces actions modifient l'emplacement du curseur et déclenchent l'événement Sur nouvelle sélection.
  - les modifications de contenu effectuées par programmation, à l'exception des commandes simulant une action utilisateur.
  Cet événement permet de contrôler les actions utilisateur afin, par exemple, d'interdire de coller un texte trop volumineux, de filtrer certains caractères ou d'empêcher le couper d'un champ de mot de passe.
- Sur nouvelle sélection : lorsqu'il est appliqué à un champ ou une variable de texte (saisissable ou non), cet événement est déclenché à chaque changement de position du curseur. C'est le cas par exemple dès que l'utilisateur sélectionne du texte à l'aide de la souris ou des touches fléchées du clavier, ou saisit du texte. Ce principe permet d'appeler par exemple des commandes telles que **TEXTE SELECTIONNE**.

Objets modifiables

Les objets modifiables sont des objets ayant une source de données, dont la valeur peut être modifiée à l'aide de la souris ou du clavier, mais qui ne sont pas gérés par l'événement Sur clic. Ces objets sont les suivants :

- Tous les champs saisissables (sauf BLOB)
- Toutes les variables saisissables (sauf BLOB, pointeurs et tableaux)
- Combo-boxes
- Objets externes (pour lesquels la saisie de données est validée par le plug-in)
- Listes hiérarchiques
- List box et colonnes de list box

Ces objets reçoivent les événements Sur données modifiées. Lorsque la propriété d'événement Sur données modifiées est sélectionnée pour un de ces objets, vous pouvez détecter et gérer la modification de la valeur de la source de données à l'aide de la commande **Evenement formulaire** qui retourne Sur données modifiées. L'événement est généré dès que la variable associée à l'objet est mise à jour en interne par 4D (c'est-à-dire, en général, lorsque la zone de saisie de l'objet perd le focus).

Objets tabulables

Les objets tabulables sont ceux qui peuvent recevoir le focus lorsque vous utilisez la touche **Tab** et/ou si vous cliquez dessus. L'objet qui a le focus est celui qui reçoit les caractères saisis via le clavier et qui ne sont pas les modificateurs d'une commande de menu ou d'un objet tel qu'un bouton.

TOUS les objets sont tabulables SAUF ceux listés ci-dessous :

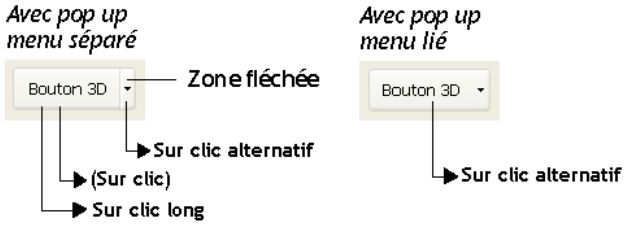
- Variables ou champs non saisissables
- Grilles de boutons
- Boutons 3D, boutons radio 3D, cases à cocher 3D
- Pop-up/listes déroulantes
- Menus déroulants hiérarchiques
- Pop-up menus Image
- Zones de défilement
- Boutons invisibles, boutons inversés, boutons radio Images
- Graphes
- Objets externes (pour lesquels la saisie de données est validée par le plug-in 4D)
- Onglets
- Séparateurs

Lorsque les événements Sur gain focus et/ou Sur perte focus sont sélectionnés pour un objet tabulable, vous pouvez détecter et gérer la modification du focus à l'aide de la commande **Evenement formulaire** qui retournera Sur gain focus ou Sur perte focus en fonction de l'événement.

Boutons 3D

Les boutons 3D autorisent la mise en place d'interfaces graphiques avancées (pour une description des boutons 3D, reportez-vous au manuel Mode Développement). Outre les événements génériques, deux événements spécifiques permettent de gérer de ces boutons :

- Sur clic long : cet événement est généré lorsqu'un bouton 3D reçoit un clic et que le bouton de la souris reste enfoncé pendant un certain laps de temps. En pratique, le délai à l'issue duquel l'événement est généré est égal au délai maximal séparant un double-clic, tel qu'il a été défini dans les préférences du système.
  Cet événement peut être généré pour tous les styles de boutons 3D, boutons radio 3D et cases à cocher 3D, à l'exception des boutons 3D "ancienne génération" (style Décalage du fond) et des zones de flèche des boutons 3D avec pop up menu (cf. ci-dessous).
  Cet événement est généralement utilisé pour afficher des pop up menus en cas de clics longs sur des boutons. L'événement Sur clic, s'il est coché, est généré si l'utilisateur relâche le bouton de la souris avant le délai du "clic long".
- Sur clic alternatif : certains styles de boutons 3D peuvent être associés à un pop up menu et afficher une flèche. Un clic sur cette flèche fait généralement apparaître un menu de sélection proposant des actions alternatives supplémentaires en rapport avec l'action principale du bouton. 4D vous permet de gérer ce type de bouton à l'aide de l'événement Sur clic alternatif. Cet événement est généré lorsque l'utilisateur clique sur la "flèche" (l'événement est généré dès que le bouton de la souris est enfoncé) :
  - si le pop up menu est de type "Séparé", l'événement est généré uniquement en cas de clic sur la zone fléchée du bouton.
  - si le pop up menu est de type "Lié", l'événement est généré en cas de clic sur n'importe quelle partie du bouton. A noter qu'avec ce type de bouton, l'événement Sur clic long ne peut pas être généré.



Les styles de boutons 3D, boutons radio 3D et cases à cocher 3D acceptant la propriété "Avec pop up menu" sont : Aucun, Bouton barre outils, Bevel, Bevel arrondi et Office XP.

List box

Plusieurs événements formulaire permettent de prendre en charge les spécificités des list box :

- Sur avant saisie : cet événement est généré juste avant qu'une cellule de list box passe en mode édition (c'est-à-dire, avant que le curseur de saisie soit affiché). Cet événement permet au développeur, par exemple, d'afficher un texte différent selon que l'utilisateur est en mode affichage ou édition.
- Sur nouvelle sélection : cet événement est généré à chaque fois que la sélection courante de lignes ou de colonnes de la list box est modifiée. Cet événement est également généré pour les listes d'enregistrements et les listes hiérarchiques.
- Sur déplacement colonne : cet événement est généré lorsqu'une colonne de list box est déplacée par l'utilisateur via le glisser-déposer. Il n'est pas généré si la colonne est glissée et déposée à son emplacement initial. La commande **LISTBOX NUMERO COLONNE DEPLACEE** permet de connaître le nouvel emplacement de la colonne.
- Sur déplacement ligne : cet événement est généré lorsqu'une ligne de list box est déplacée par l'utilisateur via le glisser-déposer. Il n'est pas généré si la ligne est glissée et déposée à son emplacement initial.
- Sur redimensionnement colonne : cet événement est généré lorsque la largeur d'une colonne de list box est modifiée par l'utilisateur. A compter de la v16, cet événement est déclenché en continu, c'est-à-dire de façon répétée durant toute la durée de l'événement, tant que la list box ou la colonne concernée est redimensionnée. Ce redimensionnement peut être effectué manuellement, mais peut également être la conséquence du changement de taille du formulaire (via une action utilisateur ou la commande **REDIMENSIONNER FENETRE FORMULAIRE**) lorsqu'il entraîne le

redimensionnement de la list box et de ses colonnes.
Note : L'événement Sur redimensionnement colonne n'est pas généré lorsqu'une colonne factice est redimensionnée (pour plus d'informations sur les colonnes factices, veuillez vous reporter à la section **Thème Redimensionnement**).

- Sur déployer et Sur contracter : ces événements sont générés lorsqu'une ligne de list box hiérarchique est déployée ou contractée.
- Sur clic entête : cet événement est généré lorsqu'un clic se produit sur l'en-tête d'une colonne de list box. Dans ce cas, la commande **Self** permet de connaître l'en-tête de colonne sur laquelle le clic s'est produit. L'événement Sur clic est généré lorsqu'un clic droit (Windows) ou un Ctrl+clic (Mac OS) se produit sur une colonne ou un en-tête de colonne. Vous pouvez tester le nombre de clics effectués par l'utilisateur à l'aide de la commande **Nombre de clics**.
  Si la propriété **Triable** a été cochée pour la list box, il est possible d'autoriser ou non le tri standard sur la colonne en passant la valeur 0 ou -1 dans la variable $0 :
  - Si $0 vaut 0, le tri standard est effectué.
  - Si $0 vaut -1, le tri standard n'est pas effectué et l'en-tête n'affiche pas la flèche de tri. Le développeur peut toutefois générer un tri des colonnes sur des critères personnalisés à l'aide des commandes de gestion des tableaux de 4D.
  Si la propriété **Triable** n'a pas été cochée pour la list box, la variable $0 n'est pas utilisée.
- Sur clic pied : cet événement est disponible pour l'objet list box ou colonne de list box. Il est généré lorsqu'un clic se produit dans la zone de pied de la list box ou de la colonne de list box. Dans ce cas, la commande **OBJET Lire pointeur** retourne un pointeur vers la variable de pied ayant reçu le clic. L'événement est généré pour les clics gauche et droit. Vous pouvez tester le nombre de clics effectués par l'utilisateur à l'aide de la commande **Nombre de clics**.
- Sur après tri : cet événement est généré juste après qu'un tri standard ait été effectué (i.e. il n'est pas généré si $0 retourne -1 dans l'événement Sur clic entête). Ce mécanisme est utile pour conserver le sens du dernier tri effectué par l'utilisateur. Dans cet événement, la commande **Self** retourne un pointeur sur la variable d'en-tête de la colonne ayant été triée.
- Sur action suppression : cet événement est généré à chaque fois que l'utilisateur tente de supprimer la ou les ligne(s) sélectionnée(s) en appuyant sur une touche de suppression (**Suppr** ou **Ret. Arr.**) ou en sélectionnant la commande **Supprimer** dans le menu **Edition**. L'événement est disponible uniquement au niveau de l'objet list box. A noter que la génération de l'événement est la seule action effectuée par 4D : le programme ne supprime aucune ligne. Il appartient au développeur de prendre en charge la suppression et éventuellement l'affichage préalable d'un message d'alerte.
- Sur défilement (nouveauté v15) : cet événement est généré dès que l'utilisateur fait défiler les lignes ou les colonnes de la list box. L'événement est uniquement généré lorsque le défilement est le résultat d'une action utilisateur : activation des barres et/ou des curseurs de défilement, utilisation de la roulette de la souris ou du clavier. Il n'est donc pas généré lorsque le défilement est provoqué par l'exécution de la commande **OBJET FIXER DEFILEMENT**.
  Cet événement est généré après tous les autres événements liés à l'action de défilement (Sur clic, Sur après frappe clavier, etc.). L'événement est généré uniquement dans la méthode objet (et non dans la méthode formulaire). Reportez-vous à l'exemple 15.
- Sur clic alternatif (nouveauté v15) : cet événement est généré dans les colonnes de list box de type tableau d'objets, lorsque l'utilisateur clique sur un bouton d'ellipse de widget (attribut "alternateButton"). Pour plus d'informations, reportez-vous à la section **Utiliser des tableaux objets dans les colonnes (4D View Pro)**.

Deux événements génériques sont également utilisables dans le contexte d'une list box de type "sélection" :

- Sur ouverture corps : cet événement est généré lorsqu'un enregistrement est sur le point d'être affiché dans le formulaire détaillé associé à la list box (et avant que ce formulaire soit ouvert).
- Sur fermeture corps : généré lorsqu'un enregistrement affiché dans le formulaire détaillé associé à la list box est sur le point d'être refermé (que l'enregistrement ait été modifié ou non).

Listes hiérarchiques

Outre les événements génériques, plusieurs événements formulaires spécifiques permettent de prendre en charge les actions utilisateurs effectuées sur les listes hiérarchiques :

- Sur nouvelle sélection : cet événement est généré à chaque fois que la sélection dans la liste hiérarchique est modifiée à la suite d'un clic ou de la frappe d'une touche au clavier.
  Cet événement est également généré dans les objets list box et les listes d'enregistrements.
- Sur déployer : cet événement est généré à chaque fois qu'un élément de la liste hiérarchique a été déployé via un clic ou une touche du clavier.
- Sur contracter : cet événement est généré à chaque fois qu'un élément de la liste hiérarchique a été contracté via un clic ou une touche du clavier.
- Sur action suppression : cet événement est généré à chaque fois que l'utilisateur tente de supprimer le ou les élément(s) sélectionné(s) en appuyant sur une touche de suppression (**Suppr** ou **Ret. Arr.**) ou en sélectionnant la commande **Supprimer** dans le menu **Edition**. A noter que la génération de l'événement est la seule action effectuée par 4D : le programme ne supprime aucun élément. Il appartient au développeur de prendre en charge la suppression et éventuellement l'affichage préalable d'un message d'alerte (cf. exemple).

Ces événements ne sont pas mutuellement exclusifs. Ils peuvent être générés les uns après les autres pour une liste hiérarchique :
- Suite à la frappe d'une touche clavier (dans l'ordre) :

Evénement	Contexte
Sur données modifiées	Un élément était en édition
Sur déployer / Sur contracter	Ouverture/fermeture de sous-liste à l'aide des touches fléchées -> ou <-
Sur nouvelle sélection	Sélection d'un nouvel élément
Sur clic	Activation de la liste par le clavier

- Suite à un clic souris (dans l'ordre) :

Evénement	Contexte
Sur données modifiées	Un élément était en édition
Sur déployer / Sur contracter	Ouverture/fermeture de sous-liste via un clic sur l'icône de déploiement/contraction ou bien Double-clic sur une sous-liste non éditable
Sur nouvelle sélection	Sélection d'un nouvel élément
Sur clic / Sur double clic	Activation de la liste par un clic ou un double-clic

Variables et champs image

- L'événement formulaire Sur défilement est généré dès que l'utilisateur fait défiler une image à l'intérieur de la zone (variable ou champ) qui la contient. Il est possible de faire défiler le contenu d'une zone image lorsque la taille de la zone est inférieure à son contenu et que le format d'affichage est "**tronqué (non centré)**". Pour plus d'informations sur ce point, reportez-vous au paragraphe **Formats image**.
  L'événement est généré lorsque le défilement a pour origine une action utilisateur, quelle que soit cette action : utilisation des curseurs ou des barres de défilement, de la molette de la souris ou des touches de défilement du clavier (pour plus d'informations sur les commandes de défilement accessibles au clavier, reportez-vous à la section **Barres de défilement**). Il n'est donc pas généré lorsque l'image défile à la suite de l'exécution de la

commande **OBJET FIXER DEFILEMENT**. Cet événement est généré après tous les autres événements liés à l'action de défilement (Sur clic, Sur après frappe clavier, etc.). Il est généré uniquement dans la méthode objet (et non dans la méthode formulaire). Reportez-vous à l'exemple 14.

- (Nouveauté v16) L'événement Sur relâchement bouton est généré lorsque l'utilisateur vient de relâcher le bouton de la souris alors qu'il était en train d'effectuer un glisser dans une zone image (champ ou variable). Cet événement est utile, par exemple, lorsque vous voulez permettre à l'utilisateur de déplacer, redimensionner ou dessiner des objets dans une zone SVG.
Lorsque l'événement Sur relâchement bouton est généré, vous pouvez récupérer les coordonnées locales de l'emplacement où le bouton de la souris a été relâché. Ces coordonnées sont retournées dans les **Variables système** **MouseX** et **MouseY**. Les coordonnées relatives sont exprimées en pixels par rapport au coin supérieur gauche de l'image (0,0).
Lorsque vous utilisez cet événement, il est également nécessaire d'appeler la commande **Attente relachement clic** pour traiter les cas où le "gestionnaire d'état" du formulaire est désynchronisé, c'est-à-dire lorsque l'événement Sur relâchement bouton n'est jamais reçu après un clic. C'est le cas par exemple lorsqu'une boîte de dialogue d'alerte s'affiche au-dessus du formulaire alors que le bouton de la souris n'a pas encore été relâché. Pour plus d'informations et un exemple d'utilisation de l'événement Sur relâchement bouton, veuillez vous référer à la description de la commande **Attente relachement clic**.

  Note : Si la propriété "Glissable" est cochée pour l'objet image, l'événement Sur relâchement bouton n'est jamais généré.

Sous-formulaires

Un objet conteneur de sous-formulaire (objet inclus dans le formulaire parent, contenant une instance de sous-formulaire) prend en charge les événements suivants :

- Sur chargement et Sur libération : respectivement ouverture et fermeture du sous-formulaire (ces événements doivent également avoir été activés au niveau du formulaire parent pour être pris en compte). A noter que ces événements sont générés avant ceux du formulaire parent. A noter également que, conformément aux principes de fonctionnement des événements formulaire, si le sous-formulaire est placé sur une page autre que la page 0 ou 1, ces événements ne sont générés qu'au moment de l'affichage/la fermeture de la page (et non du formulaire).
- Sur validation : validation de la saisie dans le sous-formulaire.
- Sur données modifiées : la variable associée à l'objet conteneur du sous-formulaire a été modifiée.
- Sur gain focus et Sur perte focus : le conteneur de sous-formulaire vient de recevoir ou de perdre le focus. Ces événements sont générés dans la méthode de l'objet sous-formulaire lorsqu'ils sont cochés. Ils sont envoyés à la méthode formulaire du sous-formulaire, ce qui permet par exemple de gérer l'affichage de boutons de navigation dans le sous-formulaire en fonction du focus. A noter que les objets du sous-formulaire peuvent eux-mêmes recevoir le focus.
- Sur modif variable liée : cet événement spécifique est généré dans le contexte de la méthode formulaire du sous-formulaire dès qu'une valeur est affectée à la variable associée au sous-formulaire dans le formulaire parent (même si la même valeur est réaffectée), et si le sous-formulaire appartient à la page formulaire courante ou à la page 0. Ce principe permet de mettre à jour le contenu du sous-formulaire à la suite de la modification de la variable associée au conteneur du sous-formulaire dans le formulaire parent. Pour plus d'informations sur la gestion des sous-formulaires, reportez-vous au manuel *Mode Développement*.

Note : Il est possible de définir tout type d'événement personnalisé pouvant être généré dans un sous-formulaire via la commande **APPELER CONTENEUR SOUS FORMULAIRE**. Cette commande permet d'appeler la méthode de l'objet conteneur et de lui passer un code d'événement.

Note : Les événements Sur clic et Sur double clic générés dans le sous-formulaire sont reçus en premier lieu par la méthode formulaire du sous-formulaire puis par la méthode formulaire du formulaire hôte.

Zones Web

Sept événements formulaires spécifiques sont disponibles pour les zones Web :

- Sur début chargement URL : cet événement est généré au début du chargement d'un nouvel URL dans la zone Web. La variable "URL" associée à la zone Web vous permet de connaître l'URL en cours de chargement.
  Note : L'URL en cours de chargement est différent de l'URL courant (reportez-vous à la description de la commande **WA Lire URL courant**).

- Sur chargement ressource URL : cet événement est généré à chaque chargement d'une nouvelle ressource (image, frame, etc.) dans la page Web courante. La variable "Progression du chargement" associée à la zone vous permet de connaître l'état courant du chargement.
- Sur fin chargement URL : cet événement est généré lorsque toutes les ressources de l'URL courant ont été chargées. Vous pouvez appeler la commande **WA Lire URL courant** afin de connaître l'URL chargé.
- Sur erreur chargement URL : cet événement est généré lorsqu'une erreur a été détectée au cours du chargement d'un URL. Vous pouvez appeler la commande **WA LIRE DERNIERE ERREUR URL** afin d'obtenir des informations sur l'erreur.
- Sur filtrage URL : cet événement est généré lorsque le chargement d'un URL a été bloqué par la zone Web du fait d'un filtre mis en place via la commande **WA FIXER FILTRES URL**. Vous pouvez alors connaître l'URL bloqué à l'aide de la commande **WA Lire dernier URL filtre**.
- Sur ouverture lien externe : cet événement est généré lorsque le chargement d'un URL a été bloqué par la zone Web et que l'URL a été ouvert avec le navigateur courant du système, du fait d'un filtre mis en place via la commande **WA FIXER FILTRES LIENS EXTERNES**. Vous pouvez alors connaître l'URL bloqué à l'aide de la commande **WA Lire dernier URL filtre**.
- Sur refus ouverture fenêtre : cet événement est généré lorsque l'ouverture d'une fenêtre pop up a été bloquée par la zone Web. En effet, les zones Web 4D ne permettent pas l'ouverture de fenêtres pop up. Vous pouvez alors connaître l'URL bloqué à l'aide de la commande **WA Lire dernier URL filtre**.

Exemple 1

L'exemple suivant montre l'utilisation de l'événement Sur validation pour affecter automatiquement la date lorsque l'enregistrement est modifié :

```
  ` Méthode d'un formulaire
Au cas ou
  ` ...
   :(Evenement formulaire=Sur validation)
      [LaTable]Date de modification:=Date du jour
Fin de cas
```

Exemple 2

Dans l'exemple suivant, la gestion complète d'un menu déroulant (initialisation, clics et relâchement de l'objet) est placée dans la méthode de l'objet :

```
// Méthode objet du menu déroulant taTaille
```

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TABLEAU TEXTE(taTaille;3)
      taTaille{1}:="Petit"
      taTaille{2}:="Moyen"
      taTaille{3}:="Grand"
   :(Evenement formulaire=Sur clic)
      Si(taTaille#0)
         ALERTE("Vous avez choisi la taille "+taTaille{taTaille}+".")
      Fin de si
   :(Evenement formulaire=Sur libération)
      EFFACER VARIABLE(taTaille)
Fin de cas
```

Exemple 3

L'exemple suivant montre comment, dans une méthode objet, gérer et valider l'opération de glisser-déposer à partir d'un champ qui n'accepte que des images.

```
  ` Méthode objet du champ Image [LaTable]uneImage
Au cas ou
   :(Evenement formulaire=Sur glisser)
  ` On est en train de glisser-déposer un objet et la souris est au-dessus d'un champ
  ` Obtenir les informations sur l'objet source
      PROPRIETES GLISSER DEPOSER($vpObjetSource;$vlElémentSource;$lProcessSource)
  ` Notez que nous n'avons pas besoin de tester le numéro de process source
  ` pour la méthode objet exécutée parce qu'elle est dans le même process
      $vlTypeDonnées:=Type($vpSrcObject->)
  ` Les données source sont-elles une image (champ, variable ou tableau) ?
      Si(($vlTypeDonnées=Est une image)|($vlTypeDonnées=Est un tableau image))
  ` Accepter l'opération
         $0:=0
      Sinon
  ` Sinon, refuser l'opération
         $0:=-1
      Fin de si
   :(Evenement formulaire=Sur déposer)
  ` Les données source ont été déposées sur l'objet, donc nous avons besoin de les copier dans l'objet.
  ` Obtenir les informations sur l'objet source
      PROPRIETES GLISSER DEPOSER($vpObjetSource;$vlElémentSource;$lProcessSource)
      $vlTypeDonnées:=Type($vpSrcObject->)
      Au cas ou
  ` L'objet source est un champ ou une variable de type Image
         :($vlTypeDonnées=Est une image)
  ` Est-ce que l'objet source est dans le même process (dans la même fenêtre et le même
            formulaire)?
            Si($lProcessSource=Numero du process courant)
  ` Copier la valeur source
               [LaTable]uneImage:=$vpObjetSource->
            Sinon
  ` Sinon, est-ce que l'objet source est une variable ?
               Si(Est une variable($vpObjetSource))
  ` Obtenir la valeur du process source
                  LIRE VARIABLE PROCESS($lProcessSource;$vpObjetSource->;$vgImageGlissée)
                  [LaTable]uneImage:=$vgImageGlissée
               Sinon
  ` Sinon, utiliser APPELER PROCESS pour obtenir la valeur du champ du process source
               Fin de si
            Fin de si
  ` L'objet source est un tableau de type Image
         :($vlTypeDonnées=Est un tableau image)
  ` Est-ce que l'objet source est dans le même process (dans la même fenêtre et le même
            formulaire)?
            Si($lProcessSource=Numero du process courant)
  ` Copier la valeur source
               [LaTable]uneImage:=$vpSrcObject->{$vlElémentSource}
            Sinon
  ` Sinon, obtenir la valeur du process source
               LIRE VARIABLE PROCESS($lProcessSource;$vpObjetSource
               ->{$vlElémentSource};$vgImageGlissée)
               [LaTable]uneImage:=$vgImageGlissée
            Fin de si
      Fin de cas
Fin de cas
```

Note : Pour d'autres exemples sur la gestion des événements Sur glisser et Sur déposer, référez-vous aux exemples de la commande **PROPRIETES GLISSER DEPOSER**.

Exemple 4

L'exemple suivant est une méthode formulaire générique. Elle fait apparaître chacun des événements qui peuvent survenir lorsqu'un formulaire est utilisé comme formulaire sortie :

```
  ` Méthode formulaire d'un formulaire sortie
$vpFormTable:=Table du formulaire courant
Au cas ou
  ` ...
   :(Evenement formulaire=Sur entête)
  ` La zone en-tête va être imprimée ou affichée
      Au cas ou
         :(Avant selection($vpFormTable->))
  ` Le code pour la première rupture d'en-tête doit être placé ici
         :(Niveau=1)
  ` Le code pour la rupture d'en-tête niveau 1 doit être placé ici
         :(Niveau=2)
  ` Le code pour la rupture d'en-tête niveau 2 doit être placé ici
  ` ...
      Fin de cas
   :(Evenement formulaire=Sur impression corps)
  ` Un enregistrement va être imprimé
  ` Le code pour chaque enregistrement doit être placé ici
   :(Evenement formulaire=Sur impression sous total)
  ` Une rupture va être imprimée
      Au cas ou
         :(Niveau=0)
  ` Le code pour la rupture 0 doit être placé ici
         :(Niveau=1)
  ` Le code pour la rupture 1 doit être placé ici
  ` ...
      Fin de cas
   :(Evenement formulaire=Sur impression pied de page)
      Si(Fin de selection($vpFormTable->))
  ` Le code pour le dernier pied de page doit être placé ici
      Sinon
  ` Le code pour le pied de page doit être placé ici
      Fin de si
Fin de cas
```

Exemple 5

L'exemple suivant montre une méthode formulaire générique qui gère les événements pouvant survenir dans un formulaire sortie quand il s'affiche à l'aide de **VISUALISER SELECTION** ou **MODIFIER SELECTION**. Dans un but informatif, elle affiche l'événement dans la barre de titre de la fenêtre.

```
  ` Une méthode formulaire exemple
Au cas ou
   :(Evenement formulaire=Sur chargement)
      $vaEvénement:="Le formulaire va être affiché"
   :(Evenement formulaire=Sur libération)
      $vaEvénement:="Le formulaire sortie vient de se fermer et va disparaître de l'écran"
   :(Evenement formulaire=Sur affichage corps)
      $vaEvénement:="Affichage de l'enregistrement n°"+Chaine(Numero dans selection([LaTable]))
   :(Evenement formulaire=Sur menu sélectionné)
      $vaEvénement:="Une commande de menu a été sélectionnée"
   :(Evenement formulaire=Sur entête)
      $vaEvénement:="L'en-tête va être imprimé ou affiché"
   :(Evenement formulaire=Sur clic)
      $vaEvénement:="On a cliqué sur un enregistrement"
   :(Evenement formulaire=Sur double clic)
      $vaEvénement:="On a double-cliqué sur un enregistrement"
   :(Evenement formulaire=Sur ouverture corps)
      $vaEvénement:="On a double-cliqué sur l'enregistrement n°"+Chaine(Numero dans selection([LaTable]))
   :(Evenement formulaire=Sur fermeture corps)
      $vaEvénement:="Retour au formulaire sortie"
   :(Evenement formulaire=Sur activation)
      $vaEvénement:="La fenêtre du formulaire passe au premier plan"
   :(Evenement formulaire=Sur désactivation)
      $vaEvénement:="La fenêtre du formulaire n'est plus au premier plan"
   :(Evenement formulaire=Sur menu sélectionné)
      $vaEvénement:="Une ligne de menu a été sélectionnée"
   :(Evenement formulaire=Sur appel extérieur)
      $vaEvénement:="Un appel extérieur a été reçu"
   Sinon
      $vaEvénement:="Que se passe-t-il ? L'événement n°"+Chaine(Evenement formulaire)
Fin de cas
CHANGER TITRE FENETRE($vaEvénement)
```

Exemple 6

Pour des exemples de gestion des événements Sur avant frappe clavier et Sur après frappe clavier, référez-vous aux exemples des commandes **Lire texte edite**, **Frappe clavier** et **FILTRER FRAPPE CLAVIER**.

Exemple 7

L'exemple suivant montre comment traiter de la même manière les clics et double-clics dans une zone de défilement :

```
   ` Méthode objet pour la zone de défilement taChoix
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TABLEAU TEXTE(taChoix;...)
   ` ...
      taChoix:=0
   :((Evenement formulaire=Sur clic)|(Evenement formulaire=Sur double clic))
      Si(taChoix#0)
   ` On a cliqué sur un élément, faire quelque chose
   ` ...
      Fin de si
   ` ...
Fin de cas
```

Exemple 8

L'exemple suivant montre comment traiter les clics et double-clics de manière différente (notez l'utilisation de l'élément zéro pour conserver la valeur de l'élément sélectionné) :

```
   // Méthode objet pour la zone de défilement taChoix
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TABLEAU TEXTE(taChoix;...)
   // ...
      taChoix:=0
      taChoix{0}:="0"
   :(Evenement formulaire=Sur clic)
      Si(taChoix#0)
         Si(taChoix#Num(taChoix))
   // On a cliqué sur un élément, faire quelque chose
   // ...
   // Sauvegarder l'élément nouvellement sélectionné pour la prochaine fois
            taChoix{0}:=Chaine(taChoix)
         Fin de si
      Sinon
         taChoix:=Num(taChoix{0})
      Fin de si
   :(Evenement formulaire=Sur double clic)
      Si(taChoix#0)
   // On a double-cliqué sur un élément, faire quelque chose
      Fin de si
   // ...
Fin de cas
```

Exemple 9

L'exemple suivant montre comment maintenir une zone contenant du texte à partir d'une méthode formulaire à l'aide des événements Sur gain focus et Sur perte focus :

```
   ` Méthode formulaire [Contacts];"Entrée"
Au cas ou
   :(Evenement formulaire=Sur chargement)
      C_TEXTE(vtZoneEtat)
      vtZoneEtat:=""
   :(Evenement formulaire=Sur gain focus)
      RESOUDRE POINTEUR(Objet focus;$vsNomVar;$vlNumTable;$vlNumChamp)
      Si(($vlNumTable#0) & ($vlNumChamp#0))
         Au cas ou
            :($vlNumChamp=1) ` Champ nom
               vtZoneEtat:="Saisissez le nom du contact, il sera automatiquement mis en majuscules."
   ` ...
            :($vlNumChamp=10) ` Champ code postal
               vtZoneEtat:="Saisissez un code postal, il sera automatiquement vérifié et validé."
   ` ...
         Fin de cas
      Fin de si
   :(Evenement formulaire=Sur perte focus)
```

```
      vtZoneEtat:=""
   ` ...
Fin de cas
```

Exemple 10

L'exemple suivant montre comment traiter l'événement de fermeture de fenêtre avec un formulaire utilisé pour l'entrée des données :

```
   ` Méthode pour un formulaire entrée
$vpFormulaireTable:=Table du formulaire courant
Au cas ou
   ` ...
   :(Evenement formulaire=Sur case de fermeture)
      Si(Enregistrement modifie($vpFormulaireTable->))
         CONFIRMER("Cet enregistrement a été modifié. Voulez-vous sauvegarder les modifications ?")
         Si(OK=1)
            VALIDER
         Sinon
            NE PAS VALIDER
         Fin de si
      Sinon
         NE PAS VALIDER
      Fin de si
   ` ...
Fin de cas
```

Exemple 11

L'exemple suivant montre comment mettre en majuscules un champ Texte ou Alphanumérique chaque fois que la valeur est modifiée :

```
   ` Méthode objet pour [Contacts]Prénom
Au cas ou
   ` ...
   :(Evenement formulaire=Sur données modifiées)
      [Contacts]Prénom:=Majusc(Sous chaine([Contacts]Prénom;1;1))+Minusc(Sous chaine([Contacts]Prénom;2))
   ` ...
Fin de cas
```

Exemple 12

L'exemple suivant montre comment mettre en majuscules un champ Texte ou Alphanumérique chaque fois que la valeur est modifiée :

```
   ` Méthode objet pour [Contacts]Prénom
Au cas ou
   ` ...
   :(Evenement formulaire=Sur données modifiées)
      [Contacts]Prénom:=Majusc(Sous chaine([Contacts]Prénom;1;1))+Minusc(Sous chaine([Contacts]Prénom;2))
   ` ...
Fin de cas
```

Exemple 13

L'exemple suivant propose une manière de gérer une action de suppression dans une liste hiérarchique :

```
   ... //méthode de la liste hiérarchique
:($Event=Sur action suppression)
 TABLEAU ENTIER LONG($itemsArray;0)
 $Ref:=Elements selectionnes(<>HL;$itemsArray;*)
 $n:=Taille tableau($itemsArray)

 Au cas ou
    :($n=0)
       ALERTE("Pas d'élément sélectionné")
       OK:=0
    :($n=1)
        CONFIRMER("Voulez-vous supprimer cet élément ?")
    :($n>1)
       CONFIRMER("Voulez-vous supprimer ces éléments ?")
 Fin de cas

 Si(OK=1)
    Boucle($i;1;$n)
       SUPPRIMER DANS LISTE(<>HL;$itemsArray{$i};*)
    Fin de boucle
```

```
Fin de si
```

Exemple 14

Dans cet exemple, l'événement formulaire Sur défilement permet de synchroniser l'affichage de deux images dans un formulaire. Le code suivant est ajouté dans la méthode de l'objet "satellite" (champ image ou variable image) :

```
Au cas ou
   :(Evenement formulaire=Sur défilement)
         // on relève la position de l'image de gauche
      OBJET LIRE DEFILEMENT(*;"satellite";vPos;hPos)
         // on l'applique à l'image de droite
      OBJET FIXER DEFILEMENT(*;"plan";vPos;hPos;*)
Fin de cas
```

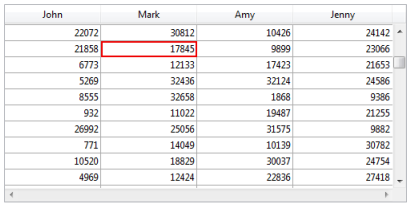
Résultat :

Exemple 15

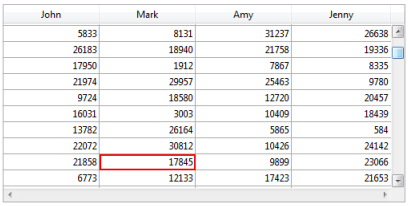
Vous souhaitez dessiner un rectangle rouge autour de la cellule sélectionnée d'une list box, et vous voulez que le rectangle se déplace si l'utilisateur fait défiler verticalement la list box. Dans la méthode objet de la list box, vous pouvez écrire :

```
Au cas ou

   :(Evenement formulaire=Sur clic)
      LISTBOX LIRE POSITION CELLULE(*;"LB1";$col;$raw)
      LISTBOX LIRE COORDONNEES CELLULE(*;"LB1";$col;$raw;$x1;$y1;$x2;$y2)
      OBJET FIXER VISIBLE(*;"RedRect";Vrai)  //initialiser rectangle rouge
      OBJET FIXER COORDONNEES(*;"RedRect";$x1;$y1;$x2;$y2)

   :(Evenement formulaire=Sur défilement)
      LISTBOX LIRE POSITION CELLULE(*;"LB1";$col;$raw)
      LISTBOX LIRE COORDONNEES CELLULE(*;"LB1";$col;$raw;$x1;$y1;$x2;$y2)
      OBJET LIRE COORDONNEES(*;"LB1";$xlb1;$ylb1;$xlb2;$ylb2)
 //tenir compte de la hauteur de l'entête pour ne pas que le rectangle empiète dessus
      $toAdd:=LISTBOX Lire hauteur entetes(*;"LB1")
      Si($ylb1+$toAdd<$y1) & ($ylb2>$y2) //si nous sommes dans la list box
 //pour simplifier, on ne tient compte que des en-têtes
 //mais il faudrait également gérer le clipping horizontal
 //ainsi que les barres de défilement
         OBJET FIXER VISIBLE(*;"RedRect";Vrai)
         OBJET FIXER COORDONNEES(*;"RedRect";$x1;$y1;$x2;$y2)
      Sinon
         OBJET FIXER VISIBLE(*;"RedRect";Faux)
      Fin de si

Fin de cas
```

En résultat, le rectangle rouge suit bien le défilement de la list box :



John	Mark	Amy	Jenny
22072	30812	10426	24142
21858	17845	9899	23066
6773	12133	17423	21653
5269	32436	32124	24586
8555	32658	1868	9386
932	11022	19487	21255
26992	25056	31575	9882
771	14049	10139	30782
10520	18829	30037	24754
4969	12424	22836	27418



John	Mark	Amy	Jenny
5833	8131	31237	26638
26183	18940	21758	19336
17950	1912	7867	8335
21974	29957	25463	9780
9724	18580	12720	20457
16031	3003	10409	18439
13782	26164	5865	584
22072	30812	10426	24142
21858	17845	9899	23066
6773	12133	17423	21653

FIXER MINUTEUR

FIXER MINUTEUR (tickCount)

Paramètre	Type		Description
tickCount	Entier long	⇨	Nombre de ticks ou -1 = Déclenchement dès que possible

Description

La commande **FIXER MINUTEUR** permet d'activer l'événement formulaire Sur minuteur et de fixer, pour le process et le formulaire courants, le nombre de ticks (1 tick = 1/60ème de seconde) entre chaque événement formulaire Sur minuteur.

Note : Pour plus d'informations sur cet événement formulaire, reportez-vous à la description de la commande **Evenement formulaire**.

Si elle est appelée dans un contexte autre que l'affichage d'un formulaire, cette commande ne fait rien.

Note : Lorsque la commande **FIXER MINUTEUR** est exécutée dans le contexte d'un sous-formulaire (méthode formulaire du sous-formulaire), l'événement Sur minuteur est généré dans le sous-formulaire et non au niveau du formulaire parent.

Si vous passez -1 dans le paramètre *tickCount*, la commande activera l'événement formulaire Sur minuteur "dès que possible", autrement dit dès que l'application 4D rendra la main au gestionnaire d'événements. Ce principe permet notamment de s'assurer qu'un formulaire soit entièrement affiché avant de démarrer un traitement (fluidité de l'application).

Pour inactiver par programmation le déclenchement de l'événement formulaire Sur minuteur, appelez de nouveau la commande **FIXER MINUTEUR** en passant 0 dans le paramètre *nbTicks*.

Exemple

Vous souhaitez que, lorsqu'un formulaire est affiché à l'écran, un bip soit émis toutes les trois secondes. Pour cela, écrivez dans la méthode du formulaire :

```
Si(Evenement formulaire=Sur chargement)
   FIXER MINUTEUR(60*3)
Fin de si
...
Si(Evenement formulaire=Sur minuteur)
   BEEP
Fin de si
```

Nombre de clics

Nombre de clics -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Nombre de clics consécutifs

Description

La commande **Nombre de clics** retourne, dans le contexte d'un événement clic, le nombre de fois que l'utilisateur a cliqué de manière répétée avec le même bouton de la souris. Typiquement, la commande retourne 2 pour un double-clic.

Cette commande vous permet notamment de détecter des double-clics dans les en-têtes ou les pieds des list box, et également de gérer des séquences de triple-clics ou plus.

Chaque clic avec un bouton de la souris génère un événement clic séparé. Par exemple, si un utilisateur effectue un double-clic, un événement est généré pour le premier clic, dans lequel **Nombre de clics** retourne 1 ; puis un autre événement est généré pour le second clic, dans lequel **Nombre de clics** retourne 2.

Cette commande peut uniquement être appelée dans le contexte de l'événement formulaire Sur clic, Sur clic entête ou Sur clic pied. Par conséquent, il est nécessaire de vérifier en mode Développement que l'événement correspondant a bien été sélectionné dans les propriétés du formulaire et/ou pour l'objet concerné.

Lorsque les deux événements formulaire Sur clic et Sur double clic sont activés, la séquence suivante est retournée par **Nombre de clics** :

- 1 dans l'événement Sur clic
- 2 dans l'événement Sur double clic
- 2+n dans l'événement Sur clic

Exemple 1

La structure de code suivante peut être placée dans un en-tête de list box pour gérer les clics simples et les double-clics :

```
Au cas ou
   :(Evenement formulaire=Sur clic entête)
      Au cas ou
         :(Nombre de clics=1)
            ... //simple clic
         :(Nombre de clics=2)
            ... //double clic
      Fin de cas
Fin de cas
```

Exemple 2

Les libellés ne sont pas saisissables mais ils peuvent le devenir après un triple-clic. Si vous souhaitez permettre aux utilisateurs de modifier les libellés, vous pouvez écrire la méthode objet suivante :

```
Si(Evenement formulaire=Sur clic)
   Au cas ou
      :(Nombre de clics=3)
         OBJET FIXER SAISISSABLE(*;"Label";Vrai)
         EDITER ELEMENT(*;"Label")
   Fin de cas
Fin de si
```

_o_Pendant

_o_Pendant -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si le cycle d'exécution est pendant

Note de compatibilité

L'utilisation de cette ancienne fonction générique de cycle d'exécution est déconseillée. Il est recommandé d'utiliser la fonction **Evenement formulaire** et de tester les événements spécifiques qu'elle retourne, comme Sur clic.

Fenêtres

- Gestion des fenêtres
- Types de fenêtres
- AFFICHER BARRE OUTILS
- AFFICHER FENETRE
- CACHER BARRE OUTILS
- CACHER FENETRE
- CHANGER COORDONNEES FENETRE
- CHANGER TITRE FENETRE
- Chercher fenetre
- CONVERTIR COORDONNEES
- COORDONNEES FENETRE
- Creer fenetre
- Creer fenetre formulaire
- DEPLACER FENETRE
- EFFACER FENETRE
- Fenetre formulaire courant
- Fenetre premier plan
- Fenetre suivante
- FERMER FENETRE
- Hauteur barre outils
- LISTE FENETRES
- MAXIMISER FENETRE
- MINIMISER FENETRE
- Process de la fenetre
- REDESSINER FENETRE
- REDIMENSIONNER FENETRE FORMULAIRE
- Titre fenetre
- Type fenetre
- *_o_Creer fenetre externe*
- Types de fenêtres (compatibilité)

Gestion des fenêtres

Le rôle des fenêtres est d'afficher des informations pour l'utilisateur. Trois actions principales nécessitent l'affichage d'une fenêtre : la saisie de données, l'affichage de données et l'affichage de messages destinés à l'utilisateur.

Il y a toujours au moins une fenêtre ouverte à l'écran. Si nécessaire, des barres de défilement sont ajoutées, afin de permettre à l'utilisateur de faire défiler le contenu d'un formulaire lorsque celui-ci est plus grand que la fenêtre. En mode Développement, cette fenêtre sera soit la fenêtre présentant vos enregistrements sous forme de liste (formulaire sortie), soit la fenêtre proposant un enregistrement en saisie (formulaire entrée). En mode Application, cette fenêtre sera l'écran présentant par défaut le logo de 4D.

Lorsque vous sélectionnez une commande de menu en mode Application, la fenêtre d'accueil peut être effacée et remplacée par des données lorsque vous faites appel aux commandes qui affichent des formulaires. Une fois que l'exécution de ces commandes est terminée, l'écran d'accueil apparaît de nouveau par défaut.

RefFen

Pour créer vos propres fenêtres, faites appel à la commande **Creer fenetre** ou **Creer fenetre formulaire**. Il existe différents *types de fenêtres* personnalisées. Toutes les fenêtres ouvertes par ces commandes sont référencées au moyen de l'expression **RefFen**. Une *RefFen* identifie de façon unique une fenêtre ouverte. C'est une expression de type Entier long. Toutes les commandes fonctionnant avec des fenêtres personnalisées attendent une *RefFen* comme paramètre.

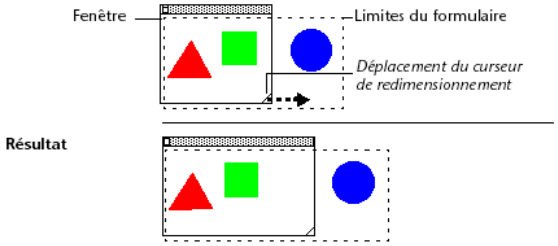
Lorsque vous n'en avez plus besoin, vous pouvez fermer une fenêtre personnalisée avec la commande **FERMER FENETRE**, ou en double-cliquant sur la case du menu Système (Windows) ou sur la case de fermeture (Mac OS), s'il y en a une.

Certaines commandes, telles que **QR ETAT** ou **IMPRIMER ETIQUETTES** ouvrent leurs propres fenêtres et les placent au premier plan.

Si vous démarrez un nouveau process et n'ouvrez pas de fenêtre au démarrage de la méthode process, 4D en créera une automatiquement avec le type par défaut, dès qu'il sera nécessaire d'afficher un formulaire.

Bords pousseurs

Les bords droits et bas des fenêtres sont par défaut des séparateurs "pousseurs". Cela signifie que les objets se trouvant à droite ou au-dessous des limites d'une fenêtre affichée à l'écran seront automatiquement repoussés vers la droite ou vers le bas en cas d'agrandissement de la fenêtre :



Ce mécanisme permet notamment de gérer des fenêtres à volets escamotables de type Explorateur (voir l'exemple de la commande **FORM FIXER TAILLE**).

Note : Ce principe n'est pas mis en oeuvre lorsque la fenêtre comporte des barres de défilement.

Coordonnées des fenêtres et mode droite à gauche

Dans les commandes de gestion des fenêtres, les coordonnées des fenêtres sont déterminées par rapport au point d'origine généralement situé en haut à gauche de la fenêtre/de l'écran.
Toutefois, lorsque le mode "droite à gauche" est activé pour l'application, les coordonnées sont inversées et le point d'origine passe en haut à droite de la fenêtre/de l'écran. Par conséquent, dans ce mode les coordonnées horizontales manipulées par les commandes suivantes doivent être inversées :

Creer fenetre
Creer fenetre formulaire
_o_Creer fenetre externe
COORDONNEES FENETRE
CHANGER COORDONNEES FENETRE
Chercher fenetre

Note : Pour plus d'informations sur le mode "droite à gauche", reportez-vous au manuel Mode Développement et à la description de la commande **FIXER PARAMETRE BASE**.

Types de fenêtres

Présentation

Vous spécifiez le type de fenêtre à ouvrir avec **Creer fenetre formulaire** à l'aide d'une des constantes prédéfinies suivantes (thème "**Creer fenetre formulaire**") :

Constante	Type	Valeur
Form dialogue modal	Entier long	1
Form dialogue modal déplaçable	Entier long	5
Form fenêtre standard	Entier long	8
Form fenêtre pop up	Entier long	32
Form fenêtre feuille	Entier long	33
Form fenêtre barre outils	Entier long	35
Form fenêtre palette	Entier long	1984
Form avec mode plein écran Mac	Entier long	65536

Cette section illustre chaque type de fenêtre, sous Windows (à gauche) et macOS (à droite).

Fenêtres modales

Une fenêtre modale place l'utilisateur dans un état (ou “mode”) où il ne peut agir qu'à l'intérieur de la fenêtre. Tant qu'une fenêtre modale est affichée, les commandes de menu et les autres fenêtres de l'application sont inaccessibles. Pour refermer une fenêtre modale, l'utilisateur doit la valider, l'annuler ou utiliser une des options qu'elle propose. Les boîtes de dialogue d'alerte sont des exemples typiques de fenêtres modales.
Dans 4D, les fenêtres de type 1 et 5 sont modales.

Note : Une fenêtre modale reste toujours au premier plan. Par conséquent, lorsqu'une fenêtre modale appelle une fenêtre non modale, cette dernière s'affiche en arrière-plan, bien qu'elle ait été appelée postérieurement. Il ne faut donc pas effectuer ce type d'opération.
En revanche, lorsqu'une fenêtre modale appelle une autre fenêtre modale, cette dernière s'affiche au premier plan.

Dialogue modal (1)

- Peut avoir un titre : Non
- Peut comporter une case de fermeture ou un équivalent : Non
- Peut être redimensionnée : Non
- Peut être agrandie/réduite ou "zoomée" : Non
- Adaptée aux barres de défilement : Non
- Utilisation : **DIALOGUE**, **AJOUTER ENREGISTREMENT**(...;...*) ou équivalent
- Les fenêtres de ce type sont modales

Dialogue modal déplaçable (5)

- Peut avoir un titre : Oui
- Peut comporter une case de fermeture ou un équivalent : Oui
- Peut être redimensionnée : Oui
- Peut être agrandie/réduite ou "zoomée" : Oui
- Adaptée aux barres de défilement : Non
- Utilisation : **DIALOGUE**, **AJOUTER ENREGISTREMENT**(...;...*) ou équivalent
- Les fenêtres de ce type sont modales mais peuvent être déplacées

Fenêtre standard (8)

- Peut avoir un titre : Oui
- Peut comporter une case de fermeture ou un équivalent : Oui
- Peut être redimensionnée : Oui
- Peut être agrandie/réduite ou "zoomée" : Oui
- Adaptée aux barres de défilement : Oui
- Utilisation : saisie de données avec des barres de défilement, **VISUALISER SELECTION**, **MODIFIER SELECTION**, etc.

Fenêtre pop up (32)

Ce type de fenêtre dispose de propriétés avancées spécifiques :

- La fenêtre ne peut pas avoir de case de fermeture mais est automatiquement refermée avec annulation lorsque :
  - un clic se produit en-dehors de la fenêtre ;
  - la fenêtre d'arrière-plan ou la fenêtre MDI est déplacée ;
  - l'utilisateur appuie sur la touche **Echap** (ou **Esc**).
- Cette fenêtre s'affiche devant une fenêtre "parente" (elle ne doit d'ailleurs pas être utilisée comme fenêtre principale d'un process). La fenêtre d'arrière-plan n'est pas désactivée. En revanche, elle ne reçoit plus d'événement.
- Il n'est pas possible de redimensionner ou de déplacer la fenêtre à l'aide de la souris ; toutefois, lorsque cette opération est effectuée par programmation, le redessinement des éléments d'arrière-plan est optimisé.
- Utilisation : ce type de fenêtre est particulièrement adapté à la prise en charge des pop up menus associés aux boutons 3D de type "bevel" ou "barres outils".
- Limitations :
  - Il n'est pas possible d'afficher un objet pop up menu à l'intérieur d'une fenêtre de ce type.
  - Depuis la version 13 de 4D, ce type de fenêtre ne permet pas l'affichage des infobulles sous Mac OS.

Fenêtre feuille (33)

Les fenêtres feuilles (sheet windows) sont des fenêtres spécifiques de l'interface macOS. Ces fenêtres "descendent" de la barre de titre de la fenêtre principale via une animation et s'affichent par-dessus celle-ci. Elles sont automatiquement centrées dans la fenêtre principale. Leurs propriétés sont comparables à celles des boîtes de dialogue modales. Elles sont généralement utilisées pour effectuer une action en relation directe avec celle se déroulant dans la fenêtre principale.

- Il n'est possible de créer une fenêtre feuille sous macOS que si la dernière fenêtre ouverte est visible et de type document (formulaire).
- La commande crée une fenêtre de type 1 (Dialogue modal) au lieu du type 33 :
  - si la dernière fenêtre ouverte n'est pas visible ou n'est pas de type document,
  - sous Windows.
- Comme une fenêtre feuille doit être dessinée par-dessus un formulaire, son affichage est repoussé dans l'événement Sur chargement du premier formulaire chargé dans la fenêtre (cf. exemple 3 de la commande **Creer fenetre formulaire**).
- Utilisation : **DIALOGUE**, **AJOUTER ENREGISTREMENT**(...;...*) ou équivalent, sous macOS (non standard sous Windows).

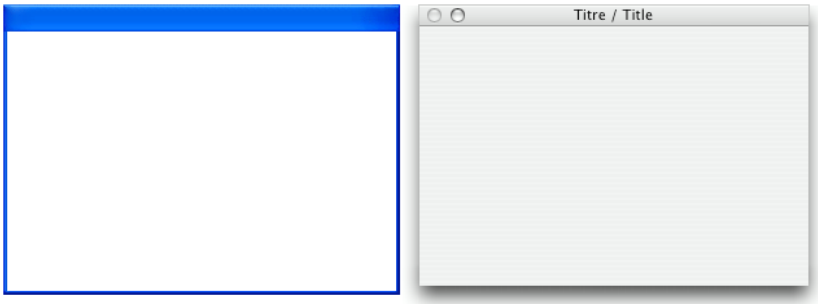
Fenêtre barre outils (35)

Une fenêtre de type "barre outils" est créée avec l'emplacement, la taille et les propriétés graphiques d'une barre d'outils, c'est-à-dire :

- La fenêtre est toujours affichée juste sous la barre de menus.
- La largeur de la fenêtre est automatiquement ajustée afin de remplir tout l'espace horizontal disponible sur le bureau (sous macOS) ou dans la fenêtre principale de 4D (sous Windows). La hauteur de la fenêtre est basée sur les propriétés du formulaire.
- La fenêtre n'a pas de bordure, elle ne peut pas être déplacée ni redimensionnée manuellement.
- Il n'est pas possible de créer simultanément plus d'une barre d'outils par process.

Barre d'outils et mode plein écran sous OS X : Si votre application affiche à la fois une fenêtre barre d'outils et une fenêtre standard qui prend en charge le mode plein écran (option Form avec mode plein écran Mac), les règles d'interface préconisent que la barre d'outils soit masquée lorsque la fenêtre standard passe en mode plein écran. Pour savoir si une fenêtre est passée en mode plein écran, il vous suffit de tester si sa hauteur est identique à celle de l'écran (cf. commande **CACHER BARRE OUTILS**).

Fenêtre palette (1984)



Ce type de fenêtre permet de générer des palettes flottantes redimensionnables ou non. Seules les options suivantes sont prises en charge :

Option	Valeur à passer sous Windows	Valeur à passer sous macOS
Non redimensionnable	-(Fenêtre palette+2)	-Fenêtre palette
Redimensionnable	-(Fenêtre palette+6)	-(Fenêtre palette+6)

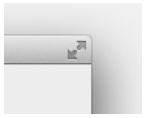
- Peut avoir un titre : Oui s'il est passé
- Peut être redimensionnée : Oui si la valeur appropriée est passée

- Utilisation : fenêtres flottantes avec **DIALOGUE** ou **VISUALISER SELECTION** (pas de saisie de données).

Note : Avec ce type de fenêtre, l'ensemble (constante + option) doit toujours être passé en valeur négative. Attention à passer par exemple -(Fenêtre palette+6) et non (-Fenêtre palette+6)

Avec mode plein écran Mac (65536)

L'option "plein écran" est disponible depuis 4D v14 sous macOS pour les fenêtres de type document. Lorsque cette option est utilisée, le bouton "Plein écran" est affiché dans l'angle supérieur droit de la fenêtre :



Lorsque l'utilisateur clique sur cette icône, la fenêtre passe en plein écran et 4D masque automatiquement la barre d'outils principale.
Pour utiliser cette option, il suffit d'ajouter la constante Form avec mode plein écran Mac au paramètre *type*. Par exemple, ce code crée une fenêtre formulaire avec bouton plein écran sous macOS :

```
$fen :=Creer fenetre formulaire([Interface];"Choix_User";Form fenêtre standard+Form avec mode plein écran Mac)
DIALOGUE([Interface];"Choix_User")
```

Note : Sous Windows, cette option n'a pas d'effet.

AFFICHER BARRE OUTILS

AFFICHER BARRE OUTILS
Ne requiert pas de paramètre

Description

La commande **AFFICHER BARRE OUTILS** permet de gérer l'affichage des barres d'outils personnalisées créées par la commande **Creer fenetre formulaire** pour le process courant.

Si une fenêtre barre d'outils a été créée par la commande **Creer fenetre formulaire** avec l'option Form fenêtre barre outils, la commande rend visible la fenêtre. Si la fenêtre barre d'outils était déjà visible ou si aucune fenêtre de ce type n'a été créée, la commande ne fait rien.

Exemple

Reportez-vous à l'exemple de la commande **CACHER BARRE OUTILS**.

AFFICHER FENETRE

AFFICHER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	⇨	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis

Description

La commande **AFFICHER FENETRE** permet d'afficher la fenêtre dont vous avez passé le numéro de référence dans *fenêtre* ou, si ce paramètre est omis, la fenêtre de premier plan du process courant.

La fenêtre doit au préalable avoir été cachée à l'aide de la commande **CACHER FENETRE**. Si la fenêtre est déjà affichée, la commande ne fait rien.

Exemple

Voir l'exemple de la commande **CACHER FENETRE**.

CACHER BARRE OUTILS

CACHER BARRE OUTILS
Ne requiert pas de paramètre

Description

La commande **CACHER BARRE OUTILS** permet de gérer l'affichage des barres d'outils personnalisées créées par la commande **Creer fenetre formulaire** pour le process courant.

Si une fenêtre barre d'outils a été créée par la commande **Creer fenetre formulaire** avec l'option Form fenêtre barre outils, la commande masque la fenêtre. Si la fenêtre barre d'outils était déjà masquée ou si aucune fenêtre de ce type n'a été créée, la commande ne fait rien

Exemple

Sous OS X, vous avez défini une barre d'outils personnalisée ainsi qu'une fenêtre standard ayant l'option Avec mode plein écran Mac. Lorsque la fenêtre standard est passée en plein écran par un utilisateur alors que la barre d'outils est affichée, vous ne voulez pas que la barre d'outils empiète sur la fenêtre plein écran.

Pour cela, dans l'événement Sur redimensionnement du formulaire de la fenêtre standard, il suffit de détecter le passage en mode plein écran et d'appeler **CACHER BARRE OUTILS** dans ce cas :

```
Au cas ou
   :(Evenement formulaire=Sur redimensionnement)
      COORDONNEES FENETRE($left;$top;$right;$bottom)
      Si(Hauteur ecran=($bottom-$top))
         CACHER BARRE OUTILS
      Sinon
         AFFICHER BARRE OUTILS
      Fin de si
Fin de cas
```

CACHER FENETRE

CACHER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	⇨	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis

Description

La commande **CACHER FENETRE** permet de masquer la fenêtre dont vous avez passé le numéro de référence dans *fenêtre* ou, si ce paramètre est omis, la fenêtre de premier plan du process courant. Cette commande vous permet, par exemple, dans un process comportant plusieurs fenêtres, de ne conserver à l'écran que la fenêtre active.

La fenêtre disparaît de l'écran mais reste ouverte. Vous pouvez continuer à lui appliquer par programmation tout traitement supporté par les fenêtres 4D.

Pour réafficher une fenêtre masquée par **CACHER FENETRE** :

- Utilisez la commande **AFFICHER FENETRE** et passez-lui le numéro de référence de la fenêtre.
- Ou bien, utilisez la page **Process** de l'Explorateur d'exécution. Sélectionnez le process dans lequel la fenêtre est gérée (process de gestion de la fenêtre ou Process principal) puis cliquez sur le bouton **Montrer**.

Si vous souhaitez cacher toutes les fenêtres d'un process, utilisez la commande **CACHER PROCESS**.

Exemple

Cet exemple est la méthode d'un bouton placé dans un formulaire entrée. Ce bouton ouvre une boîte de dialogue dans une nouvelle fenêtre du même process. Vous souhaitez masquer les autres fenêtres du process (un formulaire de saisie et une palette d'outils) afin de ne présenter que la boîte de dialogue. Une fois que celle-ci a été validée, vous réaffichez les fenêtres du process.

```
   ` Méthode objet du bouton "Informations"

CACHER FENETRE(Saisie) ` Masquer la fenêtre de saisie
CACHER FENETRE(Palette) ` Masquer la palette
$Infos:=Creer fenetre(20;100;500;400;8) ` Créer la fenêtre d'informations
... ` Placer ici les instructions nécessaires au remplissage et à la gestion du dialogue
FERMER FENETRE($Infos) ` Fermer le dialogue
AFFICHER FENETRE(Saisie)
AFFICHER FENETRE(Palette) ` Réafficher les autres fenêtres du process
```

CHANGER COORDONNEES FENETRE

CHANGER COORDONNEES FENETRE (gauche ; haut ; droite ; bas {; fenêtre}{; *})

Paramètre	Type		Description
gauche	Entier long	⇒	Coordonnée gauche de l'intérieur de la fenêtre
haut	Entier long	⇒	Coordonnée supérieure de l'intérieur de la fenêtre
droite	Entier long	⇒	Coordonnée droite de l'intérieur de la fenêtre
bas	Entier long	⇒	Coordonnée inférieure de l'intérieur de la fenêtre
fenêtre	RefFen	⇒	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process si ce paramètre est omis
*	Opérateur	⇒	Si omis (défaut) = passer la fenêtre au premier plan Si passé = ne pas changer le plan de la fenêtre

Description

La commande **CHANGER COORDONNEES FENETRE** modifie les coordonnées globales de la fenêtre dont vous avez passé le numéro de référence *RefFen* dans le paramètre *fenêtre*. Si la fenêtre n'existe pas, la commande ne fait rien.

Si vous omettez le paramètre *fenêtre*, **CHANGER COORDONNEES FENETRE** s'applique à la fenêtre de premier plan du process courant.

Ces coordonnées doivent être exprimées relativement au coin supérieur gauche de la zone de contenu de la fenêtre d'application (sous Windows) ou de l'écran principal (sous Mac OS). Les coordonnées décrivent le rectangle correspondant à la zone de contenu de la fenêtre (en excluant les barres de titres et les bordures).

Attention : Utilisez cette commande avec précaution, car vous pouvez déplacer une fenêtre en-dehors des limites de la fenêtre principale (sous Windows) ou de l'écran (sous Mac OS). Pour éviter cela, vous pouvez utiliser des fonctions telles que **Largeur ecran** et **Hauteur ecran** pour bien vérifier les nouvelles coordonnées de la fenêtre.

Par défaut, l'exécution de la commande fait automatiquement passer au premier plan la fenêtre désignée par le paramètre *fenêtre* (lorsque ce paramètre est utilisé). Vous pouvez inactiver ce fonctionnement en passant * en dernier paramètre. Dans ce cas, la commande ne modifie pas le plan initial de la fenêtre (coordonnée "z").

Cette commande n'affecte pas les objets du formulaire. Si la fenêtre contient un formulaire, les objets du formulaire ne sont pas déplacés ou redimensionnés par la commande (quelles que soient leurs propriétés). Seule la fenêtre est modifiée. Pour modifier une fenêtre de formulaire en tenant compte de ses propriétés de redimensionnement et des objets qu'elle contient, vous devez utiliser la commande **REDIMENSIONNER FENETRE FORMULAIRE**.

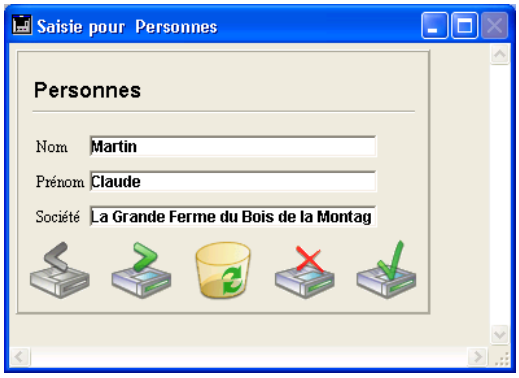
Constante	Type	Valeur

Exemple 1

Reportez-vous à l'exemple de la commande **LISTE FENETRES**.

Exemple 2

Soit la fenêtre suivante :



Après l'exécution de la ligne suivante :

```
CHANGER COORDONNEES FENETRE(100;100;300;300)
```

La fenêtre apparaît ainsi :

CHANGER TITRE FENETRE

CHANGER TITRE FENETRE (titre {; fenêtre})

Paramètre	Type		Description
titre	Chaîne	➡	Titre de la fenêtre
fenêtre	RefFen	➡	Numéro de référence de fenêtre ou Fenêtre au premier plan du process courant si ce paramètre est omis

Description

La commande **CHANGER TITRE FENETRE** remplace le titre de la fenêtre dont le numéro de référence est passé dans *fenêtre* par le texte passé dans *titre* (longueur maximale 80 caractères).

Si la fenêtre n'existe pas, **CHANGER TITRE FENETRE** ne fait rien.

Si vous omettez le paramètre *fenêtre*, **CHANGER TITRE FENETRE** remplace le titre de la fenêtre de premier plan du process courant.

Note : En mode Développement, 4D définit automatiquement les titres des fenêtres — par exemple “Saisie pour table1” est affiché lorsque vous passez en saisie de données. Si vous changez le titre d'une fenêtre du mode Développement, il est probable que 4D le remplacera par la suite. En revanche, en mode Application, 4D ne modifie pas le titre des fenêtres.

Exemple

Vous effectuez une saisie dans un formulaire et vous cliquez sur un bouton qui déclenche une longue opération (par exemple une modification par programmation des enregistrements liés affichés dans un sous-formulaire). Vous pouvez afficher des informations sur la progression des opérations dans le titre de la fenêtre :

```
   ` Méthode objet du bouton bAnalyse
Au cas ou
   :(Evenement formulaire=Sur clic)
   ` Sauvegarde du titre courant de la fenêtre dans une variable
      $vsTitreCour:=Titre fenetre
   ` Commencer l'opération longue
      DEBUT SELECTION([Lignes facture])
      Boucle($vlRecord;1;Enregistrements trouves([Lignes facture]))
         FAIRE QUELQUE CHOSE
   ` Afficher la progression
         CHANGER TITRE FENETRE("Traitement de la ligne #"+Chaine($vlEnreg))
      Fin de si
   ` Remettre en place l'ancien titre de fenêtre
      CHANGER TITRE FENETRE($vsTitreCour)
Fin de cas
```

Chercher fenetre

Chercher fenetre (gauche ; haut {; partieFenêtre}) -> Résultat

Paramètre	Type		Description
gauche	Entier long	➡	Coordonnée globale gauche
haut	Entier long	➡	Coordonnée globale supérieure
partieFenêtre	Entier long	⬅	3 si une fenêtre est "touchée", 0 sinon
Résultat	RefFen	↩	Numéro de référence de fenêtre

Description

La commande **Chercher fenetre** retourne (s'il existe) le numéro de référence de la première fenêtre "touchée" par le point dont vous passez les coordonnées dans *gauche* et *haut*.

Ces coordonnées doivent être exprimées relativement au coin supérieur gauche de la zone de contenu (l'intérieur) de la fenêtre d'application (sous Windows) ou de l'écran principal (sous Mac OS).

Le paramètre *partieFenêtre* retourne 3 si une fenêtre est touchée, et 0 sinon (**Note de compatibilité :** à compter de 4D v14, les constantes du thème **Chercher fenetre** sont obsolètes).

CONVERTIR COORDONNEES

CONVERTIR COORDONNEES (coordX ; coordY ; depuis ; vers)

Paramètre	Type		Description
coordX	Variable entier long	➡	Coordonnée horizontale d'un point (initiale)
		⬅	Coordonnée horizontale d'un point (convertie)
coordY	Variable entier long	➡	Coordonnée verticale d'un point (initiale)
		⬅	Coordonnée verticale d'un point (convertie)
depuis	Entier long	➡	Système de coordonnées d'origine
vers	Entier long	➡	Système de coordonnées dans lequel convertir le point

Description

La commande **CONVERTIR COORDONNEES** permet de convertir les coordonnées (x;y) d'un point depuis un système de coordonnées vers un autre système de coordonnées. Les systèmes de coordonnées pris en charge sont les formulaires (ainsi que les sous-formulaires), les fenêtres et l'écran. Par exemple, vous pouvez utiliser cette commande pour obtenir les coordonnées, dans le formulaire principal, d'un objet appartenant à un sous-formulaire. Ce principe facilite notamment la création de menus contextuels à des emplacements personnalisés.

Dans *coordX* et *coordY*, passez des variables contenant les coordonnées (x;y) du point que vous voulez convertir. Après exécution de la commande, ces variables contiendront les valeurs converties.

Dans le paramètre *depuis*, passez le système d'origine dans lequel sont exprimées les coordonnées du point, et dans le paramètre *vers*, passez le système de coordonnées dans lequel elles doivent être converties. Chaque paramètre peut avoir pour valeur l'une des constantes suivantes, présentes dans le thème "**Fenêtre**" :

Constante	Type	Valeur	Comment
XY Ecran	Entier long	3	L'origine est le coin supérieur de l'écran principal (comme pour la commande **COORDONNEES ECRAN**)
XY Fenêtre courante	Entier long	2	L'origine est le coin supérieur gauche de la fenêtre courante
XY Fenêtre principale	Entier long	4	Windows : L'origine est le coin supérieur gauche de la fenêtre principale ; OS X : identique à XY Ecran
XY Formulaire courant	Entier long	1	L'origine est le coin supérieur gauche du formulaire courant

Lorsque cette commande est appelée depuis la méthode d'un sous-formulaire ou d'un objet du sous-formulaire, si l'un des sélecteurs est XY Formulaire courant, les coordonnées correspondantes sont relatives au sous-formulaire lui-même, et non celles de son formulaire parent.

Lorsque vous effectuez une conversion depuis/vers la position d'une fenêtre de formulaire (par exemple une conversion depuis les résultats de **COORDONNEES FENETRE**, ou vers des valeurs passées à **Creer fenetre formulaire**), le sélecteur XY Fenêtre principale doit être utilisé car il s'agit du système de coordonnées utilisé par les commandes de gestion des fenêtres sous Windows. Ce sélecteur peut également être utilisé dans ce but sous OS X, où il est équivalent à XY Ecran.

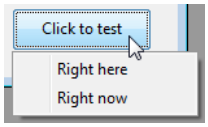
Lorsque le sélecteur *depuis* contient XY Formulaire courant et que le point à convertir est situé dans la zone de corps d'un formulaire liste, le résultat dépend du contexte d'appel de la commande :

- Si la commande est appelée dans l'événement Sur affichage corps, le point résultant est situé dans le périmètre d'affichage de l'enregistrement affiché à l'écran.
- Si la commande est appelée en-dehors de l'événement Sur affichage corps mais qu'un enregistrement est en cours de modification, le point résultant est situé dans le périmètre d'affichage de l'enregistrement en cours d'édition.
- Sinon, le point résultant est situé dans le périmètre d'affichage du premier enregistrement.

Exemple 1

Vous souhaitez afficher un pop up menu à l'angle inférieur gauche de l'objet "MonObjet" :

```
// OBJECT GET COORDINATES / OBJET LIRE COORDONNEES utilise
// le système de coordonnées du formulaire courant
// Dynamic pop up menu / Pop up menu dynamique utilise
// le système de coordonnées de la fenêtre courante
// Il faut donc convertir les valeurs
C_ENTIER LONG($left;$top;$right;$bottom)
C_TEXTE($menu)
OBJET LIRE COORDONNEES(*;"MonObjet";$left;$top;$right;$bottom)
CONVERTIR COORDONNEES($left;$bottom;XY Formulaire courant;XY Fenêtre courante)
$menu:=Creer menu
AJOUTER LIGNE MENU($menu;"Right here")
AJOUTER LIGNE MENU($menu;"Right now")
Pop up menu dynamique($menu;"";$left;$bottom)
EFFACER MENU($menu)
```



Exemple 2

Vous souhaitez créer une fenêtre pop up à l'emplacement du curseur de la souris. Sous Windows, vous devez convertir les coordonnées car **POSITION SOURIS** (avec le paramètre *) retourne des valeurs basées sur la position de la fenêtre MDI :

```
C_ENTIER LONG($mouseX;$mouseY;$mouseButtons)
C_ENTIER LONG($window)
POSITION SOURIS($mouseX;$mouseY;$mouseButtons)
CONVERTIR COORDONNEES($mouseX;$mouseY;XY Fenêtre courante;XY Fenêtre principale)
$window:=Creer fenetre formulaire("PopupWindowForm";Form fenêtre pop up;$mouseX;$mouseY)
DIALOGUE("PopupWindowForm")
FERMER FENETRE($window)
```

COORDONNEES FENETRE

COORDONNEES FENETRE (gauche ; haut ; droite ; bas {; fenêtre})

Paramètre	Type		Description
gauche	Entier long	⇐	Coordonnée gauche de l'intérieur de la fenêtre
haut	Entier long	⇐	Coordonnée supérieure de l'intérieur de la fenêtre
droite	Entier long	⇐	Coordonnée droite de l'intérieur de la fenêtre
bas	Entier long	⇐	Coordonnée inférieure de l'intérieur de la fenêtre
fenêtre	RefFen	⇒	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process si omis ou Fenêtre MDI si -1 (Windows)

Description

La commande **COORDONNEES FENETRE** retourne les coordonnées globales de la fenêtre dont vous avez passé le numéro de référence dans le paramètre *fenêtre*. Si la fenêtre n'existe pas, les variables des paramètres sont inchangées.

Si vous omettez le paramètre *fenêtre*, **COORDONNEES FENETRE** s'applique à la fenêtre de premier plan du process courant.

Les coordonnées retournées sont exprimées relativement au coin supérieur gauche de la zone de contenu de la fenêtre d'application (sous Windows) ou de l'écran principal (sous Mac OS). Les coordonnées retournent le rectangle correspondant à la zone de contenu de la fenêtre (en excluant les barres de titres et les bordures).

Note : Sous Windows, si vous passez -1 dans *fenêtre*, **COORDONNEES FENETRE** retourne les coordonnées de la fenêtre d'application (fenêtre MDI) relativement à l'écran.

Exemple

Reportez-vous à l'exemple de la commande **LISTE FENETRES**.

Creer fenetre

Creer fenetre (gauche ; haut ; droite ; bas {; type {; titre {; caseFermeture}}}) -> Résultat

Paramètre	Type		Description
gauche	Entier long	⇒	Coordonnée gauche de l'intérieur de la fenêtre
haut	Entier long	⇒	Coordonnée supérieure de l'intérieur de la fenêtre
droite	Entier long	⇒	Coordonnée droite de l'intérieur de la fenêtre
bas	Entier long	⇒	Coordonnée inférieure de l'intérieur de la fenêtre
type	Entier long	⇒	Type de la fenêtre
titre	Chaîne	⇒	Titre de la fenêtre
caseFermeture	Chaîne	⇒	Méthode à appeler en cas de double-clic sur la case du menu Système ou de clic sur la case de fermeture
Résultat	RefFen	⇐	Numéro de référence de la fenêtre

Description

Creer fenetre ouvre une nouvelle fenêtre dont les dimensions sont définies par les quatre premiers paramètres :

- *gauche* est la distance en pixels entre le côté gauche de la fenêtre de l'application et le côté intérieur gauche de la fenêtre.
- *haut* est la distance en pixels entre le haut de la fenêtre de l'application et le bord supérieur de l'intérieur de la fenêtre.
- *droite* est la distance en pixels entre le côté gauche de la fenêtre de l'application et le côté intérieur droit de la fenêtre.
- *bas* est la distance en pixels entre le haut de la fenêtre de l'application et le bord inférieur de l'intérieur de la fenêtre.

Note de compatibilité : Creer fenetre a intégré diverses options qui ont évolué au fil des versions, et n'est plus conservée que pour des raisons de compatibilité. Lorsque vous écrivez du nouveau code de gestion des fenêtres, il est vivement conseillé d'utiliser la commande **Creer fenetre formulaire**, mieux adaptée aux interfaces actuelles.

Si vous passez -1 dans *droite* et *bas*, vous indiquez à 4D qu'il faut redimensionner automatiquement la fenêtre si les conditions suivantes sont réunies :

- Vous avez conçu un formulaire et défini ses options de dimensionnement dans la fenêtre des propriétés des formulaires du mode Développement.
- Avant d'appeler **Creer fenetre** vous avez sélectionné le formulaire à l'aide de la commande **FORM FIXER ENTREE**, à laquelle vous avez passé le paramètre optionnel *.

Important : Ce dimensionnement automatique de la fenêtre n'aura lieu que si vous avez au préalable appelé la commande **FORM FIXER ENTREE** pour le formulaire que vous allez afficher dans la fenêtre, et si vous lui avez passé le paramètre optionnel *.

- Le paramètre *type* est optionnel. Il définit le type de fenêtre que vous souhaitez afficher, et correspond aux différentes fenêtres présentées dans la section **Types de fenêtres (compatibilité)** (constantes du thème **Creer fenetre**). Si le type passé est négatif, la fenêtre sera flottante. Si le type n'est pas spécifié, le type 1 est utilisé par défaut.
- Le paramètre *titre* indique le titre (optionnel) de la fenêtre.
  Si vous passez une chaîne de caractères vide ("") dans *titre*, vous indiquez à 4D d'utiliser les valeurs saisies dans la zone **Nom de la fenêtre** de la fenêtre des Propriétés du formulaire en mode Développement pour le titre du formulaire que vous allez afficher dans la fenêtre.

Important : Le nom par défaut du formulaire ne sera appliqué à la fenêtre que si vous avez appelé la commande **FORM FIXER ENTREE** pour le formulaire que vous allez afficher dans la fenêtre et si vous lui avez passé le paramètre optionnel *.

- Le paramètre *caseFermeture*, optionnel, désigne la méthode de gestion de la fermeture de la fenêtre. Si ce paramètre est passé, la case du menu Système (sous Windows) ou une case de fermeture (sous Mac OS) est ajoutée à la fenêtre. Lorsque l'utilisateur Windows double-clique sur la case du menu Système ou que l'utilisateur Mac OS clique sur la case de fermeture, la méthode passée dans *caseFermeture* est exécutée.

Note : Vous pouvez aussi gérer la fermeture à partir de la méthode du formulaire affiché dans la fenêtre pendant l'événement Sur case de fermeture. Pour plus d'informations sur ce point, reportez-vous à la commande **Evenement formulaire**.

Si plusieurs fenêtres sont ouvertes dans le même process, la dernière fenêtre créée est la fenêtre active (de premier plan) du process. Seules les informations situées dans la fenêtre active peuvent être modifiées. Toutes les autres fenêtres peuvent être visualisées. Lorsque l'utilisateur tape une touche du clavier, la fenêtre active vient toujours se placer au premier plan, si elle n'y est pas déjà.

Les formulaires sont affichés à l'intérieur de fenêtres ouvertes à l'écran. Le texte passé à la commande **MESSAGE** est également affiché dans une fenêtre.

Creer fenetre retourne une référence de fenêtre de type *RefFen*, utilisable par les commandes de gestion de fenêtres (cf. paragraphe "**RefFen**").

Exemple 1

La méthode projet suivante ouvre une fenêtre centrée dans la fenêtre principale (sous Windows) ou dans l'écran principal (sous Mac OS). Notez qu'elle accepte deux, trois ou quatre paramètres :

```
   ` Méthode projet OUVRIR FENETRE CENTREE
   ` $1 - Largeur de la fenêtre
   ` $2 - Hauteur de la fenêtre
   ` $3 - Type de la fenêtre (optionnel)
   ` $4 - Titre de la fenêtre (optionnel)
$SW:=Largeur ecran\2
$SH:=(Hauteur ecran\2)-10
$WW:=$1\2
$WH:=$2\2
Au cas ou
   :(Nombre de paramètres=2)
      Creer fenetre($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH)
   :(Nombre de paramètres=3)
      Creer fenetre($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3)
   :(Nombre de paramètres=4)
```

```
      Creer fenetre($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3;$4)
Fin de cas
```

Une fois que cette méthode projet est écrite, vous pouvez l'utiliser de la manière suivante :

```
OUVRIR FENETRE CENTREE(400;250;Dialogue simple;"Mise à jour Archives")
DIALOGUE([Table outils];"OPTIONS MAJ")
FERMER FENETRE
Si(OK=1)
  ` ...
Fin de si
```

Exemple 2

L'exemple suivant crée une fenêtre flottante comportant une case de menu système (sous Windows) ou une case de fermeture (sous Mac OS). La fenêtre est créée dans le coin supérieur droit de la fenêtre de l'application.

```
$mafenetre:=Creer fenetre(Largeur ecran-149;33;Largeur ecran-4;178;-Fenêtre palette;"";"caseFermeture")
DIALOGUE([Dialogues];"Palette de couleurs")
```

La méthode *caseFermeture* appelle la commande **NE PAS VALIDER** :

```
NE PAS VALIDER
```

Exemple 3

L'exemple suivant ouvre une fenêtre dont le titre et la taille proviennent des propriétés du formulaire affiché dans la fenêtre :

```
FORM FIXER ENTREE([Clients];"Ajout d'enregistrements";*)
$mafenetre:=Creer fenetre(10;80;-1;-1;Fenêtre standard;"")
Repeter
   AJOUTER ENREGISTREMENT([Clients])
Jusque(OK=0)
```

Rappel

Pour que la fonction **Creer fenetre** utilise automatiquement les propriétés du formulaire, vous devez avoir appelé **FORM FIXER ENTREE** avec le paramètre optionnel * et les propriétés du formulaire doivent avoir été définies en fonction de cette utilisation.

Exemple 4

Cet exemple illustre le mécanisme de “retard” d’affichage des fenêtres feuille sous Mac OS X :

```
$maFenêtre:=Creer fenetre(10;10;400;400;Fenêtre feuille)
 `A cet instant la fenêtre est créée mais reste invisible
DIALOGUE([Table];"formDial")
 `L’événement Sur chargement est généré puis la fenêtre feuille est affichée, elle "descend"
 `du dessous de la barre de titre
```

Creer fenetre formulaire

Creer fenetre formulaire ({laTable ;} nomForm {; type {; posH {; posV {; *}}}}) -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table du formulaire ou Table par défaut si ce paramètre est omis
nomForm	Chaîne	⇒	Nom du formulaire
type	Entier long	⇒	Type de la fenêtre
posH	Entier long	⇒	Position horizontale de la fenêtre
posV	Entier long	⇒	Position verticale de la fenêtre
*	Opérateur	⇒	Conserver la position et la taille précédentes de la fenêtre
Résultat	RefFen	⇐	Numéro de référence de la fenêtre

Description

La commande **Creer fenetre formulaire** crée une nouvelle fenêtre utilisant les propriétés de taille et de redimensionnement du formulaire *nomForm*, passé en paramètre.

A noter que le formulaire *nomForm* n'est pas affiché dans la fenêtre créée. Il vous appartient, si vous le souhaitez, d'afficher le formulaire (par exemple à l'aide de la commande **AJOUTER ENREGISTREMENT**).

Le paramètre optionnel *type* vous permet de spécifier un type de fenêtre. Ce paramètre doit contenir une des constantes prédéfinies suivantes, placées dans le thème "**Creer fenetre formulaire**" :

Constante	Type	Valeur
Form avec mode plein écran Mac	Entier long	65536
Form dialogue modal	Entier long	1
Form dialogue modal déplaçable	Entier long	5
Form fenêtre barre outils	Entier long	35
Form fenêtre feuille	Entier long	33
Form fenêtre palette	Entier long	1984
Form fenêtre pop up	Entier long	32
Form fenêtre standard	Entier long	8

Notes :

- La constante Form avec mode plein écran Mac doit être ajoutée à l'une des autres constantes de type.
- Pour plus d'informations sur les types de fenêtres utilisables avec **Creer fenetre formulaire**, reportez-vous à la section **Types de fenêtres**.

Par défaut, si le paramètre *type* n'est pas passé, la fenêtre créée est de type Form fenêtre standard.

Case de fermeture

Les fenêtres de type Form dialogue modal déplaçable, Form fenêtre standard et Form fenêtre palette comportent une case de fermeture. Aucune méthode n'est associée à cette case de fermeture : un clic sur la case de fermeture provoquera simplement l'annulation du formulaire, sauf si l'événement Sur case de fermeture est activé pour le formulaire, auquel cas le code associé à cet événement sera exécuté.

Contrôle de taille

Si les propriétés "Taille de la fenêtre" du formulaire *nomForm* ne sont pas fixes, la fenêtre créée peut être redimensionnée par l'utilisateur. Une case de zoom peut également être disponible, suivant le type de la fenêtre. Si la propriété **Largeur fixe** et/ou **Hauteur fixe** est cochée dans les propriétés du formulaire, la taille de la fenêtre ne pourra pas être modifiée.

Note : Certains attributs de la fenêtre créée (case de contrôle de taille, case de fermeture...) dépendent des spécifications d'interface du système d'exploitation pour le *type* choisi. Il est donc possible d'obtenir des résultats différents en fonction de la plate-forme.

Le paramètre optionnel *posH* vous permet de définir l'emplacement horizontal de la fenêtre. Vous pouvez passer dans ce paramètre une coordonnée fixe exprimée en pixels (distance entre le côté gauche de la fenêtre de l'application et le côté intérieur gauche de la fenêtre), ou l'une des constantes prédéfinies suivantes, placées dans le thème "**Creer fenetre formulaire**" :

Constante	Type	Valeur
A droite	Entier long	196608
A gauche	Entier long	131072
Centrée horizontalement	Entier long	65536

Le paramètre optionnel *posV* vous permet de définir l'emplacement vertical de la fenêtre. Vous pouvez passer dans ce paramètre une coordonnée fixe exprimée en pixels (distance entre le haut de la fenêtre de l'application et le bord supérieur de l'intérieur de la fenêtre), ou l'une des constantes prédéfinies suivantes, placées dans le thème "**Creer fenetre formulaire**" :

Constante	Type	Valeur
Centrée verticalement	Entier long	262144
En bas	Entier long	393216
En haut	Entier long	327680

Ces paramètres tiennent compte de la présence de la barre d'outils, de la barre de menus, et de la taille courante de la fenêtre de l'application (sous Windows).

Si vous passez le paramètre optionnel *, la position et la taille courantes de la fenêtre sont mémorisées au moment où elle est refermée. Lorsque la fenêtre est réouverte par la suite, elle conserve sa position et sa taille précédentes. Dans ce cas, les paramètres *posV* et *posH* ne sont utilisés que pour la première ouverture de la fenêtre.

Note : Pour rouvrir une fenêtre avec ses coordonnées par défaut lorsque le paramètre * est passé, maintenez la touche **Maj** enfoncée lors de son ouverture.

Exemple 1

L'instruction suivante ouvre une fenêtre standard avec case de fermeture automatiquement ajustée à la taille du formulaire "Entrée". La taille de fenêtre du formulaire n'est pas fixe, la fenêtre comporte donc également une case de contrôle de taille et une case de zoom :

```
$refFen:=Creer fenetre formulaire([Table1];"Entrée")
```

Exemple 2

L'instruction suivante ouvre, en haut et à gauche de l'écran, une palette flottante basée sur un formulaire projet nommé "Outils". Cette palette conservera sa précédente position à chaque nouvelle ouverture :

```
$refFen:=Creer fenetre formulaire("Outils";Form fenêtre palette;A gauche;En haut;*)
```

Exemple 3

Ce code doit être appelé alors qu'une fenêtre document est affichée, par exemple depuis un bouton de formulaire sous macOS :

```
$maFenêtre:=Creer fenetre formulaire("sheet form";Form fenêtre feuille)
 // A cet instant la fenêtre est créée mais reste invisible
DIALOGUE([Table];"formDial")
 //l'événement Sur chargement est généré puis la fenêtre feuille est affichée, elle "descend"
 //du dessous de la barre de titre
```

DEPLACER FENETRE

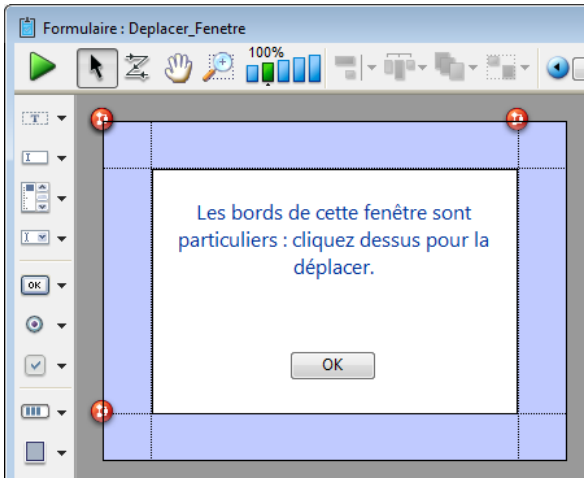
DEPLACER FENETRE
Ne requiert pas de paramètre

Description

La commande **DEPLACER FENETRE** permet de faire glisser la fenêtre dans laquelle l'utilisateur a cliqué puis de la déplacer en fonction des mouvements de la souris. Généralement, cette commande est appelée depuis la méthode d'un objet capable de répondre instantanément aux clics souris (par exemple un bouton invisible).

Exemple

Le formulaire suivant, présenté ici dans l'éditeur de formulaires, contient un fond coloré au-dessus duquel quatre boutons invisibles ont été placés (un par côté) :



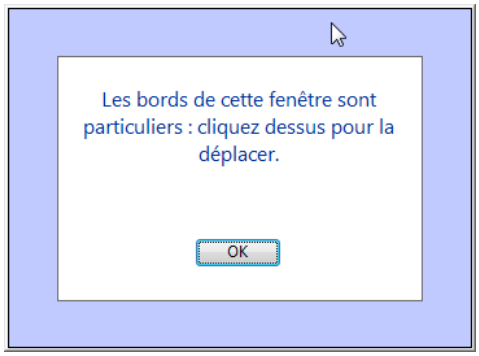
Chaque bouton est associé à la méthode suivante :

```
DEPLACER FENETRE //Commencer à faire glisser la fenêtre au premier clic
```

Après l'exécution de la méthode projet suivante :

```
$refFen:=Creer fenetre formulaire("Deplacer_Fenetre";Form dialogue modal)
DIALOGUE("Deplacer_Fenetre")
FERMER FENETRE
```

... vous obtenez une fenêtre semblable à celle-ci :



Vous pouvez la déplacer en cliquant sur les bordures.

EFFACER FENETRE

EFFACER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	⇨	Numéro de référence de fenêtre ou Fenêtre au premier plan du process courant si ce paramètre est omis

Description

La commande **EFFACER FENETRE** efface le contenu de la fenêtre dont vous avez passé la référence dans *fenêtre*.

Si vous omettez le paramètre *fenêtre*, **EFFACER FENETRE** efface le contenu de la fenêtre de premier plan du process courant.

Généralement, vous utiliserez **EFFACER FENETRE** en combinaison avec **MESSAGE** et **POSITION MESSAGE**. Dans ce cas, **EFFACER FENETRE** efface le contenu de la fenêtre et place le curseur dans son angle supérieur gauche, c'est-à-dire à la position correspondant à **POSITION MESSAGE**(0; 0).

Ne confondez pas **EFFACER FENETRE**, qui efface le contenu d'une fenêtre, et **POSITION MESSAGE**, qui supprime la fenêtre de l'écran.

Fenetre formulaire courant

Fenetre formulaire courant -> Résultat

Paramètre	Type		Description
Résultat	RefFen		Numéro de référence de la fenêtre du formulaire courant

Description

La commande **Fenetre formulaire courant** retourne la référence de la fenêtre du formulaire courant. S'il n'y a pas de fenêtre définie pour le formulaire courant, la commande retourne 0.

La fenêtre du formulaire courant peut avoir été générée automatiquement par une commande telle que **AJOUTER ENREGISTREMENT**, à la suite d'une action utilisateur ou via les commandes **Creer fenetre** ou **Creer fenetre formulaire**.

Fenetre premier plan

Fenetre premier plan {(*)} -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si omis = ignorer les fenêtres flottantes, Si spécifié = prendre en compte les fenêtres flottantes
Résultat	RefFen	⇐	Numéro de référence de fenêtre

Description

La commande **Fenetre premier plan** retourne le numéro de référence de la fenêtre actuellement située au premier plan.

Fenetre suivante

Fenetre suivante (fenêtre) -> Résultat

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre
Résultat	RefFen	↩	Numéro de référence de fenêtre

Description

La commande **Fenetre suivante** retourne le numéro de référence de la fenêtre située “derrière” la fenêtre dont vous avez passé le numéro de référence dans *fenêtre* (en fonction de l'ordre des fenêtres).

FERMER FENETRE

FERMER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	⇨	Numéro de référence de la fenêtre externe ou Fenêtre de premier plan du process si ce paramètre est omis

Description

FERMER FENETRE referme la dernière fenêtre créée à l'aide de la commande **Creer fenetre** ou **Creer fenetre formulaire** dans le process courant. S'il n'y a pas de fenêtre personnalisée ouverte, **FERMER FENETRE** ne fait rien ; la commande ne ferme pas les fenêtres système. Si **FERMER FENETRE** est appelée alors qu'un formulaire est actif dans la fenêtre, elle n'a pas d'effet non plus. Vous devez appeler **FERMER FENETRE** lorsque vous avez fini d'utiliser une fenêtre ouverte avec **Creer fenetre** ou **Creer fenetre formulaire**.

Il est inutile de passer un numéro à **FERMER FENETRE** lorsque vous l'utilisez pour refermer des fenêtres ouvertes à l'aide de la fonction **Creer fenetre** ou **Creer fenetre formulaire**. En effet, si plusieurs fenêtres ont été ouvertes par une succession d'appels à ces commandes, elles ne pourront être refermées que dans l'ordre inverse de leur création.

Si vous passez en paramètre la référence d'une zone externe créée à l'aide de la fonction **_o_Creer fenetre externe**, **FERMER FENETRE** referme la fenêtre externe. Pour plus d'informations sur les fenêtres externes, reportez-vous à la description de la fonction **_o_Creer fenetre externe**.

Exemple

L'exemple suivant ouvre une fenêtre formulaire et crée des enregistrements à l'aide de la commande **AJOUTER ENREGISTREMENT**. Une fois les enregistrements ajoutés, la fenêtre est fermée par la commande **FERMER FENETRE** :

```
FORM FIXER ENTREE([Employés];"Entrée")
$refFen:=Creer fenetre formulaire([Employés];"Entrée")
Repeter
   AJOUTER ENREGISTREMENT([Employés]) //Ajout d'un enregistrement d'employé
Jusque(OK=0) //Boucle jusqu'à ce que l'utilisateur annule
FERMER FENETRE //Fermeture de la fenêtre
```

Hauteur barre outils

Hauteur barre outils -> Résultat

Paramètre	Type		Description
Résultat	Entier long	➲	Hauteur (exprimée en pixels) de la barre d'outils ou 0 si la barre d'outils n'est pas affichée

Description

La commande **Hauteur barre outils** retourne la hauteur de la barre d'outils visible courante, exprimée en pixels. Suivant le contexte, il peut s'agir soit de la barre d'outils du mode Développement de 4D, soit d'une barre d'outils personnalisée créée avec **Creer fenetre formulaire** (la barre d'outils du mode Développement est automatiquement masquée lorsqu'une barre d'outils personnalisée est affichée).

Si aucune barre d'outils n'est affichée, la commande retourne 0.

LISTE FENETRES

LISTE FENETRES (fenêtres {; *})

Paramètre	Type		Description
fenêtres	Tableau	⇐	Tableau des numéros de référence des fenêtres
*	Opérateur	⇒	Si omis, ignorer fenêtres flottantes Si spécifié, tenir compte des fenêtres flottantes

Description

La commande **LISTE FENETRES** remplit le tableau *fenêtres* avec les numéros de référence des fenêtres actuellement ouvertes dans tous les process (process moteur et process utilisateur).

Si vous ne passez pas le paramètre optionnel *, les fenêtres flottantes sont ignorées.

Exemple

La méthode projet suivante place en "mosaïque" toutes les fenêtres ouvertes (à l'exception des fenêtres flottantes et des boîtes de dialogue) :

```
   ` Méthode projet Mosaïque

LISTE FENETRES($alWnd)
$vlLeft:=10
$vlTop:=80 ` Laissons de la place à la barre d'outils
Boucle($vlWnd;1;Taille tableau($alWnd))
   Si(Type fenetre($alWnd{$vlWnd})#Fenêtre modale)
      COORDONNEES FENETRE($vlWL;$vlWT;$vlWR;$vlWB;$alWnd{$vlWnd})
      $vlWR:=$vlLeft+($vlWR-$vlWL)
      $vlWB:=$vlTop+($vlWB-$vlWT)
      $vlWL:=$vlLeft
      $vlWT:=$vlTop
      CHANGER COORDONNEES FENETRE($vlWL;$vlWT;$vlWR;$vlWB;$alWnd{$vlWnd})
      $vlLeft:=$vlLeft+10
      $vlTop:=$vlTop+25
   Fin de si
Fin de boucle
```

Note : Cette méthode pourrait être améliorée par l'ajout de tests sur la taille de la fenêtre principale (sous Windows) ou sur la taille et l'emplacement du ou des écran(s) (sous Mac OS).

MAXIMISER FENETRE

MAXIMISER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	⇒	Numéro de référence de la fenêtre Si omis = Toutes les fenêtres (Windows) ou Fenêtre de premier plan du process courant (Mac OS)

Description

La commande **MAXIMISER FENETRE** provoque le zoom de la fenêtre dont vous avez passé le numéro de référence dans *fenêtre* ou, si ce paramètre est omis, de toutes les fenêtres de l'application (sous Windows) ou de la fenêtre de premier plan du process courant (sous Mac OS).

Cette commande produit le même effet qu'un clic sur la case de zoom d'une fenêtre de l'application 4D. Les fenêtres que vous souhaitez maximiser doivent comporter une case de zoom. Si le type de *fenêtre* n'en contient pas, la commande ne fait rien (pour plus d'informations sur ce point, reportez-vous à la section **Types de fenêtres (compatibilité)**).

Un clic ultérieur sur la case de zoom ou l'appel de la commande **MINIMISER FENETRE** provoque le retour de la fenêtre à sa taille initiale. Sous Windows, si la fenêtre a été maximisée, un clic sur la case de zoom ou l'appel de la commande **MINIMISER FENETRE** (sans paramètre) entraîne le retour à leur taille initiale de toutes les fenêtres de l'application.

Si *fenêtre* est déjà maximisée, la commande ne fait rien.

Sous Windows

La fenêtre est agrandie et s'adapte à la taille courante de la fenêtre de l'application. Si vous ne passez pas le paramètre *fenêtre*, toutes les fenêtres de l'application sont maximisées. La fenêtre maximisée est passée au premier plan.

Case de zoom ("bouton d'agrandissement") sous Windows

Dans le cas où la commande est appliquée à une fenêtre dont la taille est soumise à des contraintes (par exemple, une fenêtre formulaire) :

- Si aucune des containtes de taille n'est en conflit avec la taille cible, la fenêtre est placée dans son état "maximisé" (c'est-à-dire : elle est redimensionnée à la taille de la fenêtre MDI ("Multiple Document Interface") parente ; sa barre de titre et ses bordures sont cachées et ses boutons de contrôle - minimiser, restaurer et fermer - sont déplacés à droite de la barre de menus de l'application).
- Si au moins une des containtes de taille est en conflit (par exemple, si la largeur de la fenêtre MDI est de 100 et que la largeur maximale de la fenêtre formulaire est 80), la fenêtre n'est pas placée dans son état "maximisé", mais uniquement redimensionnée à sa taille maximale autorisée. Cette taille est définie soit par la fenêtre MDI, soit par la contrainte. Avec ce fonctionnement, l'interface reste cohérente lorsque les fenêtres avec contraintes sont redimensionnées.

Sous Mac OS

La fenêtre est agrandie de manière à afficher la totalité de son contenu. Si vous ne passez pas le paramètre *fenêtre*, la fenêtre du premier plan du process courant est maximisée.

Case de zoom sous Mac OS

- Le zoom étant calculé par rapport au contenu de la fenêtre, cette commande doit être appelée dans un contexte où ce contenu est défini, par exemple une méthode formulaire. Sinon, la commande ne fait rien.
- La fenêtre est dimensionnée à sa taille "maximale". Si la fenêtre est un formulaire dont la taille maximale a été définie dans les Propriétés du formulaire, le zoom de la fenêtre se limitera à cette taille.

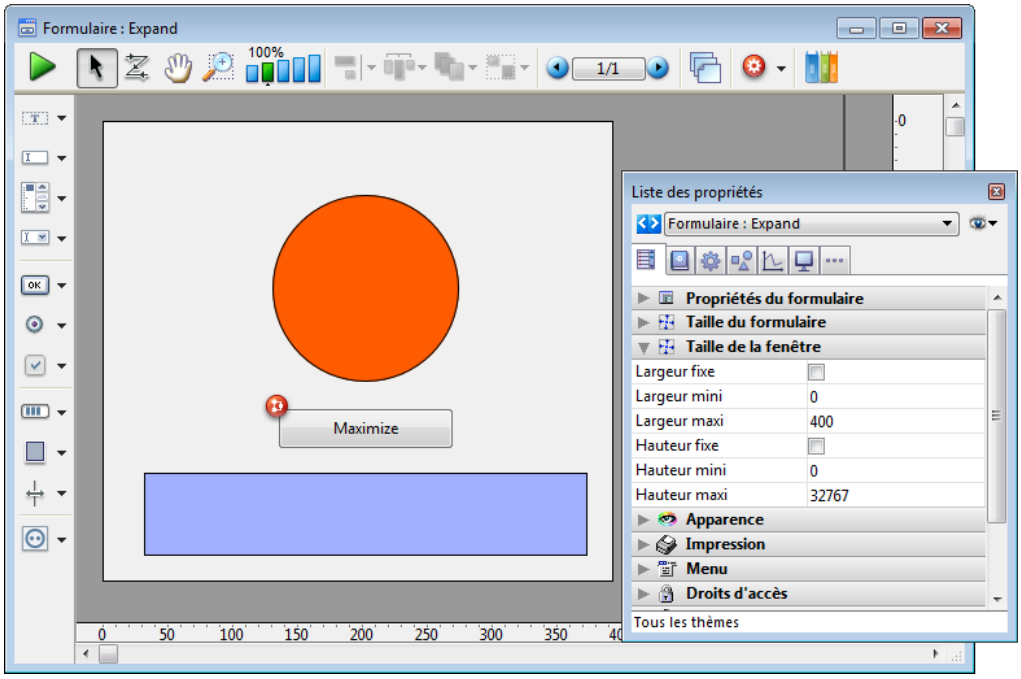
Exemple 1

Vous souhaitez que votre formulaire s'ouvre sur une fenêtre "plein écran". Pour cela, vous placez la commande **MAXIMISER FENETRE** dans la méthode du formulaire :

```
   ` Méthode formulaire

MAXIMISER FENETRE
```

Exemple 2

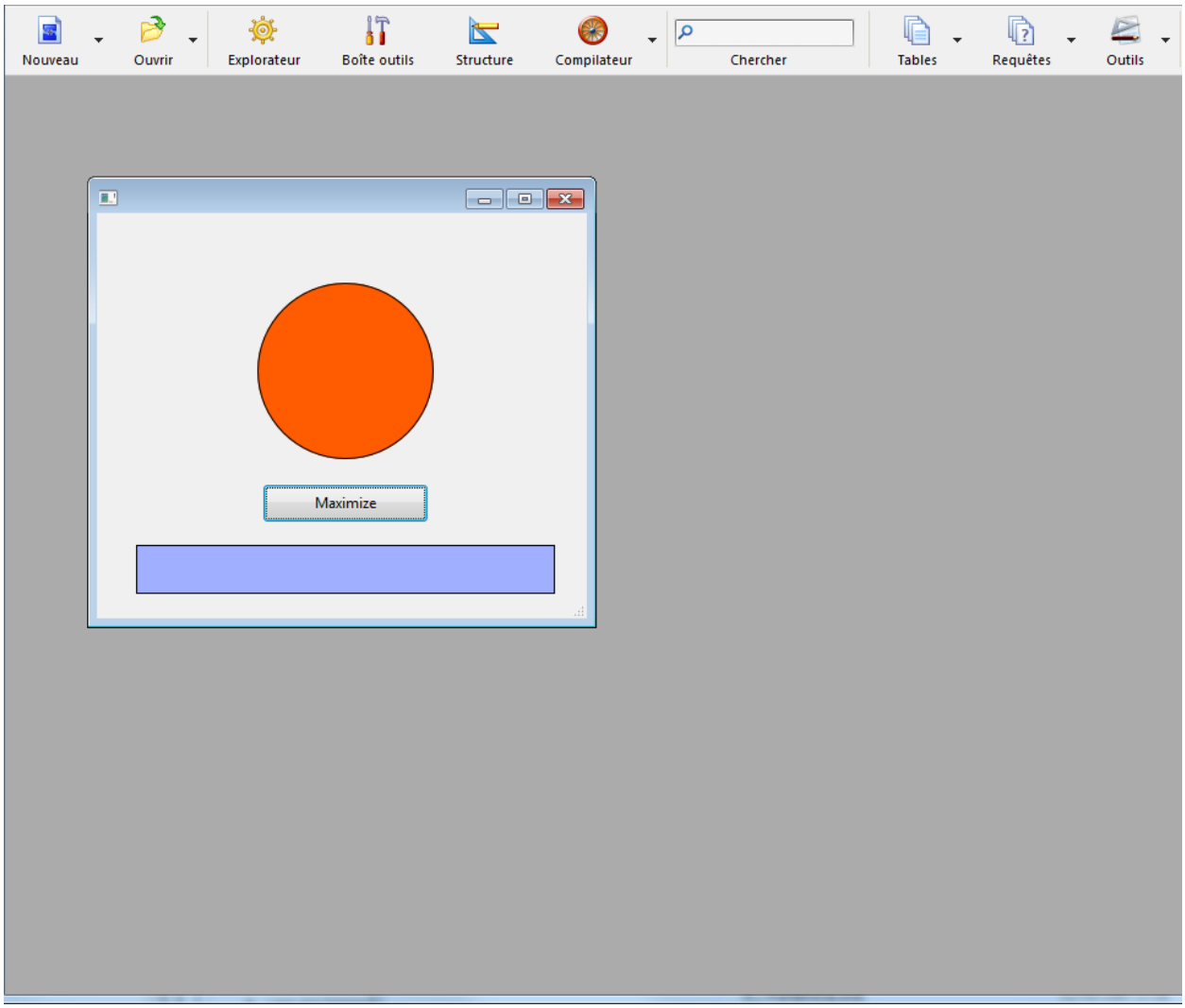
Cet exemple illustre la prise en charge des contraintes de taille sous Windows. Le formulaire suivant comporte une contrainte de taille (largeur maximale=400) :



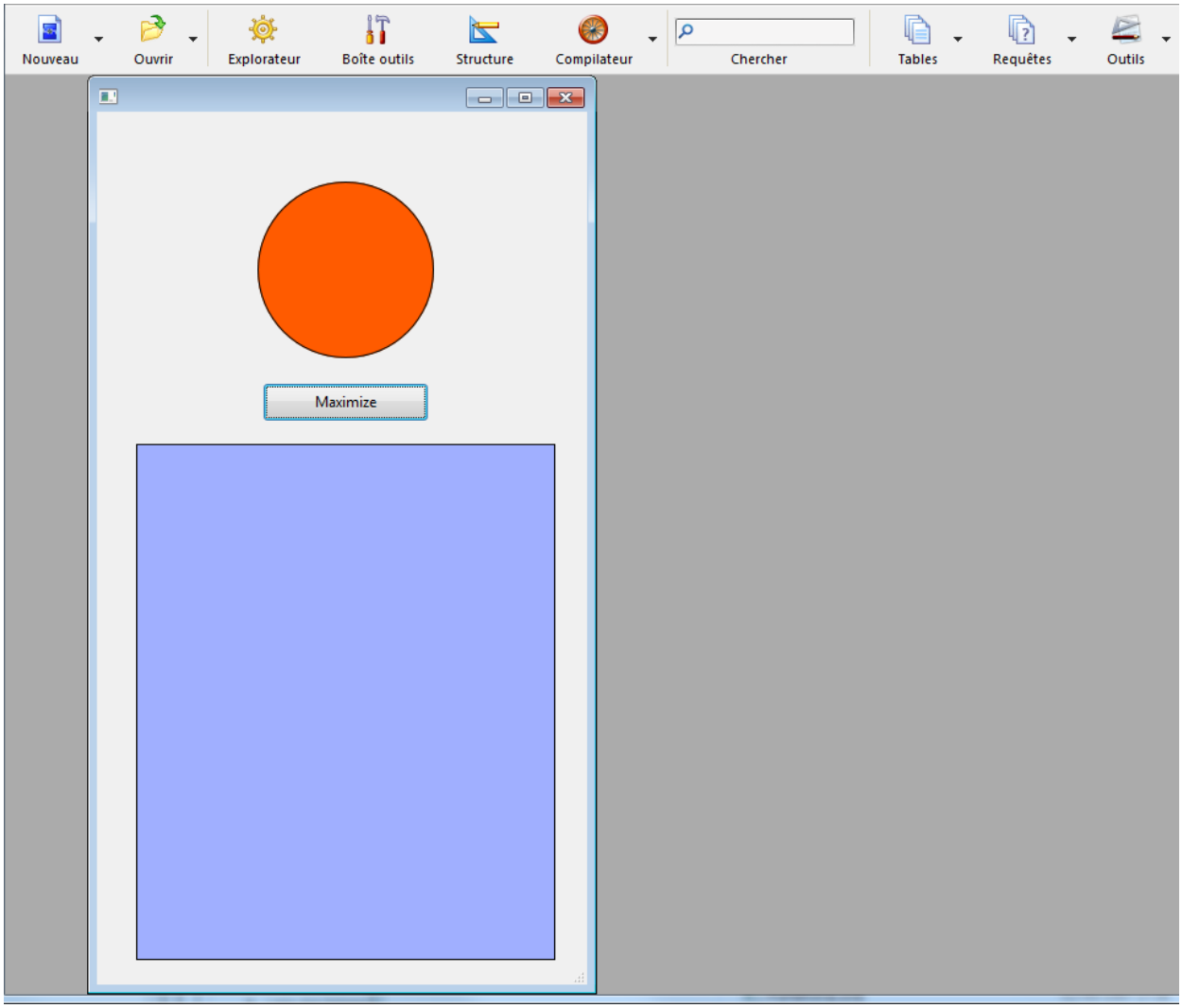
La méthode du bouton contient uniquement :

```
MAXIMISER FENETRE(Fenetre formulaire courant)
```

Dans le contexte suivant, si l'utilisateur clique sur le bouton :



... la fenêtre n'est pas placée dans son état "maximisé" ; seule sa hauteur est augmentée :


Nouveau
Ouvrir
Explorateur
Boîte outils
Structure
Compilateur
Chercher
Tables
Requêtes
Outils
Maximize

MINIMISER FENETRE

MINIMISER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	⇨	Numéro de référence de la fenêtre Si omis = Toutes les fenêtres (Windows) ou Fenêtre de premier plan du process courant (Mac OS)

Description

La commande **MINIMISER FENETRE** provoque un zoom arrière de la fenêtre dont vous avez passé le numéro de référence dans *fenêtre* ou, si ce paramètre est omis, de toutes les fenêtres de l'application (sous Windows) ou de la fenêtre de premier plan du process courant (sous Mac OS).

Cette commande produit le même effet qu'un clic sur la case de réduction d'une fenêtre de l'application 4D ayant été préalablement maximisée :

Sous Windows

La fenêtre est réduite et retrouve sa taille initiale. Si vous ne passez pas le paramètre *fenêtre*, toutes les fenêtres de l'application sont redimensionnées à leur taille initiale.

Case de réduction sous Windows

Sous Mac OS

La fenêtre est réduite et retrouve sa taille initiale. Si vous ne passez pas le paramètre *fenêtre*, la fenêtre de premier plan du process courant est minimisée.

Case de zoom/réduction sous Mac OS

Si la ou les fenêtres concernées n'ont pas été préalablement maximisées (manuellement ou à l'aide de **MAXIMISER FENETRE**), la commande ne fait rien.

De même, si le type de *fenêtre* ne comporte pas de case de zoom, la commande ne fait rien (pour plus d'informations sur ce point, reportez-vous à la section **Types de fenêtres (compatibilité)**).

Note : Ne confondez pas cette fonction avec la réduction de la fenêtre sous forme d'icône (Windows) ou dans le Dock (Mac OS), accessible par l'intermédiaire du bouton suivant :

Windows

Mac OS

Process de la fenetre

Process de la fenetre {(fenêtre)} -> Résultat

Paramètre	Type		Description
fenêtre	RefFen	➡	Numéro de référence de fenêtre
Résultat	Entier long	➲	Numéro de référence de process

Description

La commande **Process de la fenetre** retourne le numéro du process qui exécute la fenêtre dont le numéro de référence est passé dans *fenêtre*. Si la fenêtre n'existe pas, la commande retourne *0* (zéro).

Si vous omettez le paramètre *fenêtre*, **Process de la fenetre** retourne le numéro du process de la fenêtre de premier plan du process courant.

REDESSINER FENETRE

REDESSINER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	⇨	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis

Description

La commande **REDESSINER FENETRE** provoque une mise à jour du contenu de la fenêtre dont le numéro de référence est passé dans *fenêtre*.

Si vous omettez le paramètre *fenêtre*, **REDESSINER FENETRE** s'appliquera à la fenêtre de premier plan du process courant.

Note : 4D gère automatiquement les mises à jour graphiques des fenêtres à chaque fois que vous déplacez, redimensionnez ou passez au premier plan une fenêtre ainsi qu'à chaque fois que vous changez le formulaire et/ou les valeurs affiché(e)s dans une fenêtre. Cette commande est rarement utilisée.

REDIMENSIONNER FENETRE FORMULAIRE

REDIMENSIONNER FENETRE FORMULAIRE (largeur ; hauteur)

Paramètre	Type		Description
largeur	Entier long	⇒	Pixels à ajouter ou soustraire à la largeur courante de la fenêtre formulaire
hauteur	Entier long	⇒	Pixels à ajouter ou soustraire à la hauteur courante de la fenêtre formulaire

Description

La commande **REDIMENSIONNER FENETRE FORMULAIRE** permet de modifier la taille de la fenêtre du formulaire courant.

Passez dans les paramètres *largeur* et *hauteur* le nombre de pixels que vous souhaitez ajouter aux dimensions courantes de la fenêtre. Pour ne pas modifier une dimension, passez 0 dans le paramètre correspondant. Pour réduire une dimension, passez une valeur négative dans *largeur* et *hauteur*.

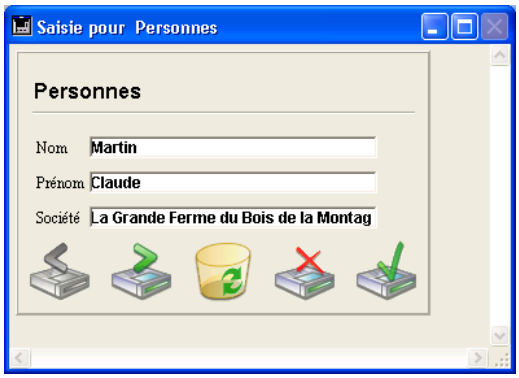
Cette commande produit exactement le même résultat qu'un redimensionnement manuel de la fenêtre à l'aide de la case de redimensionnement (si le type de fenêtre le permet). Par conséquent, la commande tient compte des propriétés de redimensionnement des objets et des contraintes de taille définies dans les propriétés du formulaire : si par exemple la commande entraîne un redimensionnement de la fenêtre supérieur à la taille maximale du formulaire, elle n'a pas d'effet.

A noter que ce fonctionnement est différent de celui de la commande **CHANGER COORDONNEES FENETRE**, qui ne tient pas compte des propriétés du formulaire ni de son contenu en cas de redimensionnement de la fenêtre.

A noter également que cette commande ne modifie pas forcément les dimensions du formulaire lui-même. Pour modifier par programmation la taille d'un formulaire, reportez-vous à la description de la commande **FORM FIXER TAILLE**.

Exemple

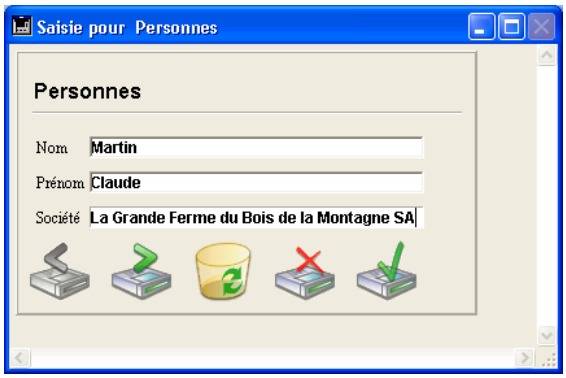
Soit la fenêtre suivante (les champs et l'encadrement ont pour propriété de redimensionnement horizontal "Agrandir") :



Après l'exécution de cette ligne :

```
REDIMENSIONNER FENETRE FORMULAIRE(25;0)
```

... la fenêtre a l'apparence suivante :



Titre fenetre

Titre fenetre {(fenêtre)} -> Résultat

Paramètre	Type		Description
fenêtre	RefFen	⇒	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis
Résultat	Chaîne	⇐	Titre de la fenêtre

Description

La commande **Titre fenetre** retourne le titre de la fenêtre dont le numéro de référence est passé dans *fenêtre*. Si la fenêtre n'existe pas, une chaîne vide est retournée.

Si vous omettez le paramètre *fenêtre*, **Titre fenetre** retourne le titre de la fenêtre de premier plan du process courant.

Exemple

Reportez-vous à l'exemple de la commande **CHANGER TITRE FENETRE**.

Type fenetre

Type fenetre {(fenêtre)} -> Résultat

Paramètre	Type		Description
fenêtre	RefFen	⇒	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis
Résultat	Entier long	⇐	Type de la fenêtre

Description

La commande **Type fenetre** retourne le type de fenêtre 4D dont vous avez passé la référence dans *fenêtre*. Si la fenêtre n'existe pas, **Type fenetre** retourne 0 (zéro).

Sinon, **Type fenetre** retourne une valeur correspondant à l'une des constantes prédéfinies suivantes (thème **Fenêtre**) :

Constante	Type	Valeur
Fenêtre externe	Entier long	5
Fenêtre flottante	Entier long	14
Fenêtre modale	Entier long	9
Fenêtre normale	Entier long	8

Si vous omettez le paramètre *fenêtre*, **Type fenetre** s'applique à la fenêtre de premier plan du process courant.

Exemple

Reportez-vous à l'exemple de la commande **LISTE FENETRES**.

_o_Creer fenetre externe

_o_Creer fenetre externe (gauche ; haut ; droite ; bas ; type ; titre ; zonePlugin) -> Résultat

Paramètre	Type		Description
gauche	Entier long	⇒	Coordonnée gauche de l'intérieur de la fenêtre
haut	Entier long	⇒	Coordonnée supérieure de l'intérieur de la fenêtre
droite	Entier long	⇒	Coordonnée droite de l'intérieur de la fenêtre
bas	Entier long	⇒	Coordonnée inférieure de l'intérieur de la fenêtre
type	Entier long	⇒	Type de la fenêtre
titre	Chaîne	⇒	Titre de la fenêtre
zonePlugin	Chaîne	⇒	Commande de zone externe
Résultat	RefFen	⇐	Numéro de référence de la fenêtre et de la zone externe

Note de compatibilité

La commande **_o_Creer fenetre externe** est déclarée obsolète dans 4D à compter de la version 16 et est conservée pour des raisons de compatibilité uniquement. A noter qu'elle n'est pas prise en charge dans les versions 64 bits de 4D.

Types de fenêtres (compatibilité)

Note de compatibilité

Certains types de fenêtres présentés dans cette section sont liés à d'anciennes versions des OS et de 4D. Ce thème ainsi que la commande **Creer fenetre** sont conservés uniquement pour des raisons de compatibilité. Lorsque vous écrivez du nouveau code de gestion des fenêtres, il est fortement recommandé d'utiliser la commande **Creer fenetre formulaire**, mieux adaptée aux interfaces actuelles.

Présentation

Vous spécifiez le type de fenêtre à ouvrir avec **Creer fenetre** à l'aide d'une des constantes prédéfinies suivantes (thème "**Creer fenetre**") :

Constante	Type	Valeur	Comment
Fenêtre standard sans zoom	Entier long	0	
Dialogue modal	Entier long	1	
Dialogue simple	Entier long	2	Utilisable en fenêtre flottante
Dialogue ombré	Entier long	3	Utilisable en fenêtre flottante
Fenêtre standard de taille fixe	Entier long	4	
Dialogue modal déplaçable	Entier long	5	Utilisable en fenêtre flottante
Fenêtre standard	Entier long	8	
Fenêtre à coins arrondis	Entier long	16	
Fenêtre pop up	Entier long	32	
Fenêtre feuille	Entier long	33	
Fenêtre feuille redim	Entier long	34	
Fenêtre palette	Entier long	1984	Utilisable en fenêtre flottante

Fenêtres flottantes

Si vous passez une de ces constantes à **Creer fenetre**, vous créez une fenêtre standard. Pour ouvrir une fenêtre flottante, passez un type de fenêtre négatif à **Creer fenetre**.
Les fenêtres flottantes ont pour caractéristique principale de rester au premier plan même si l'utilisateur clique dans une autre fenêtre du process. Les fenêtres flottantes sont généralement utilisées pour afficher des informations permanentes ou des barres d'outils.

Fenêtres modales

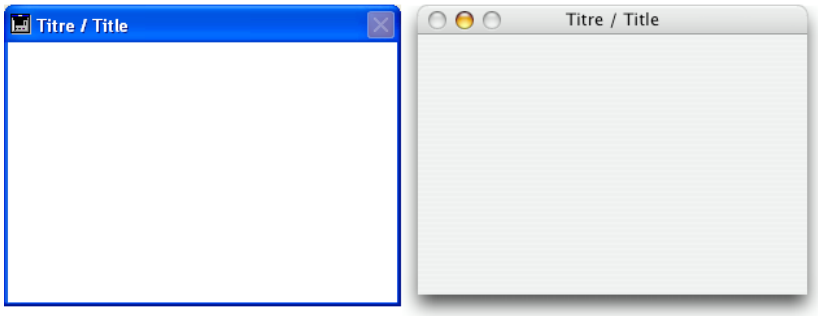
Une fenêtre modale place l'utilisateur dans un état (ou “mode”) où il ne peut agir qu'à l'intérieur de la fenêtre. Tant qu'une fenêtre modale est affichée, les commandes de menu et les autres fenêtres de l'application sont inaccessibles. Pour refermer une fenêtre modale, l'utilisateur doit la valider, l'annuler ou utiliser une des options qu'elle propose. Les boîtes de dialogue d'alerte sont des exemples typiques de fenêtres modales.
Dans 4D, les fenêtres de type 1 et 5 sont modales.

Note : Une fenêtre modale reste toujours au premier plan. Par conséquent, lorsqu'une fenêtre modale appelle une fenêtre non modale, cette dernière s'affiche en arrière-plan, bien qu'elle ait été appelée postérieurement. Il ne faut donc pas effectuer ce type d'opération.
En revanche, lorsqu'une fenêtre modale appelle une autre fenêtre modale, cette dernière s'affiche au premier plan.

Description

Voici la description de chaque type de fenêtre, sous Windows (à gauche) et Mac OS (à droite).

Fenêtre standard de taille fixe (4)

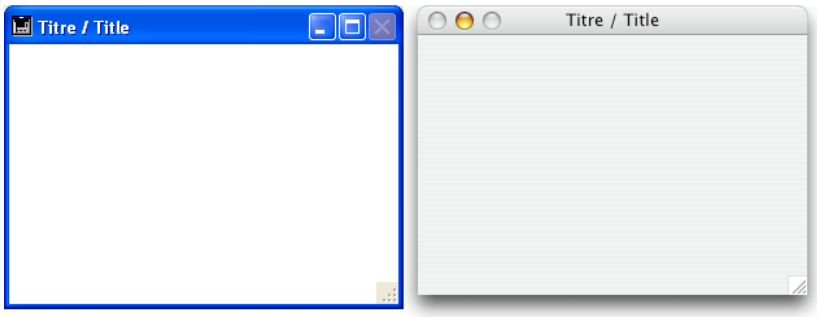


- Peut avoir un titre : Oui
- Peut comporter une case de fermeture ou un équivalent : Oui
- Peut être redimensionnée : Non sous Mac OS
- Peut être agrandie/réduite ou "zoomée" : Non
- Adaptée aux barres de défilement : Oui et Non
- Utilisation : saisie de données par **AJOUTER ENREGISTREMENT**(...;...*) ou équivalent

Dialogue modal (1)

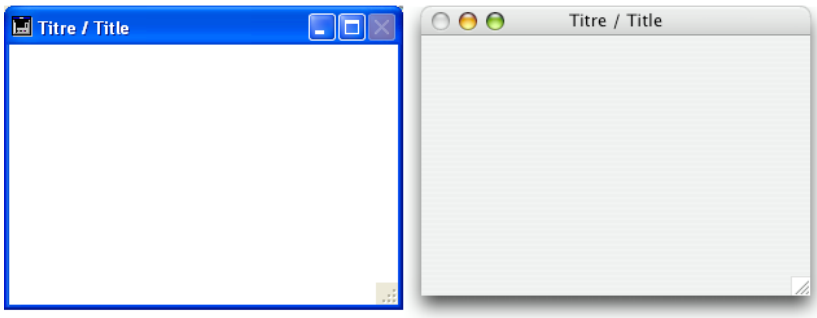
- Peut avoir un titre : Non
- Peut comporter une case de fermeture ou un équivalent : Non
- Peut être redimensionnée : Non
- Peut être agrandie/réduite ou "zoomée" : Non
- Adaptée aux barres de défilement : Non
- Utilisation : **DIALOGUE**, **AJOUTER ENREGISTREMENT**(...;...*) ou équivalent
- Les fenêtres de ce type sont modales

Fenêtre standard sans zoom (0)



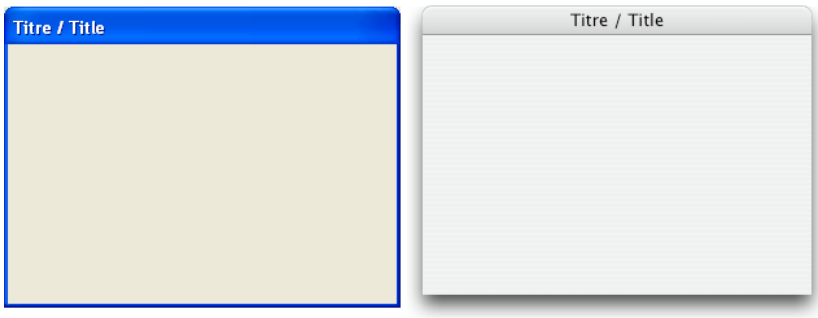
- Peut avoir un titre : Oui
- Peut comporter une case de fermeture ou un équivalent : Oui
- Peut être redimensionnée : Oui
- Peut être agrandie/réduite ou "zoomée" : Non sous Mac OS
- Adaptée aux barres de défilement : Oui
- Utilisation : saisie de données avec des barres de défilement, **VISUALISER SELECTION**, **MODIFIER SELECTION**, etc.

Fenêtre standard (8)



- Peut avoir un titre : Oui
- Peut comporter une case de fermeture ou un équivalent : Oui
- Peut être redimensionnée : Oui
- Peut être agrandie/réduite ou "zoomée" : Oui
- Adaptée aux barres de défilement : Oui
- Utilisation : saisie de données avec des barres de défilement, **VISUALISER SELECTION**, **MODIFIER SELECTION**, etc.

Dialogue modal déplaçable (5)



- Peut avoir un titre : Oui
- Peut comporter une case de fermeture ou un équivalent : Non
- Peut être redimensionnée : Non
- Peut être agrandie/réduite ou "zoomée" : Non
- Adaptée aux barres de défilement : Non

- Utilisation : **DIALOGUE**, **AJOUTER ENREGISTREMENT**(...;...*) ou équivalent
- Les fenêtres de ce type sont modales mais peuvent être déplacées et utilisées comme fenêtres flottantes

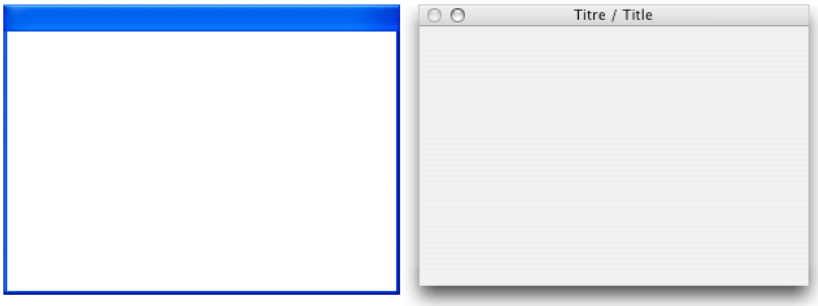
Dialogue ombré (3)

- Peut avoir un titre : Non
- Peut comporter une case de fermeture ou un équivalent : Non
- Peut être redimensionnée : Non
- Peut être agrandie/réduite ou "zoomée" : Non
- Adaptée aux barres de défilement : Non
- Utilisation : **DIALOGUE**, **AJOUTER ENREGISTREMENT**(...;...*) ou équivalent, sous Mac OS (non standard sous Windows).

Dialogue simple (2)

- Peut avoir un titre : Non
- Peut comporter une case de fermeture ou un équivalent : Non
- Peut être redimensionnée : Non
- Peut être agrandie/réduite ou "zoomée" : Non
- Adaptée aux barres de défilement : Non
- Utilisation : **DIALOGUE**, **AJOUTER ENREGISTREMENT**(...;...*) ou équivalent, sous Mac OS (non standard sous Windows).

Fenêtre palette (1984)



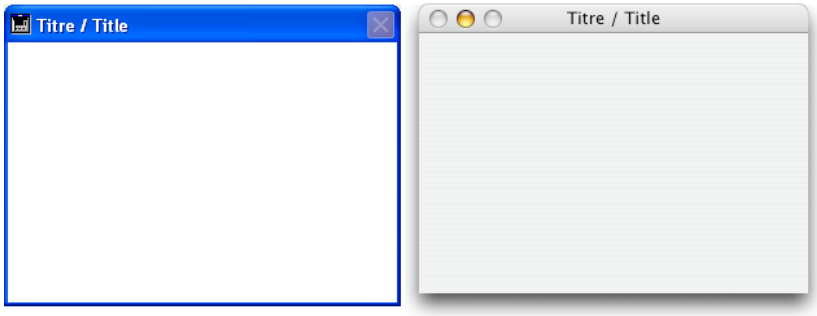
Ce type de fenêtre permet de générer des palettes flottantes redimensionnables ou non. Seules les options suivantes sont prises en charge :

Option	Valeur à passer sous Windows	Valeur à passer sous macOS
Non redimensionnable	-(Fenêtre palette+2)	-Fenêtre palette
Redimensionnable	-(Fenêtre palette+6)	-(Fenêtre palette+6)

- Peut avoir un titre : Oui s'il est passé
- Peut être redimensionnée : Oui si la valeur appropriée est passée
- Utilisation : fenêtres flottantes avec **DIALOGUE** ou **VISUALISER SELECTION** (pas de saisie de données).

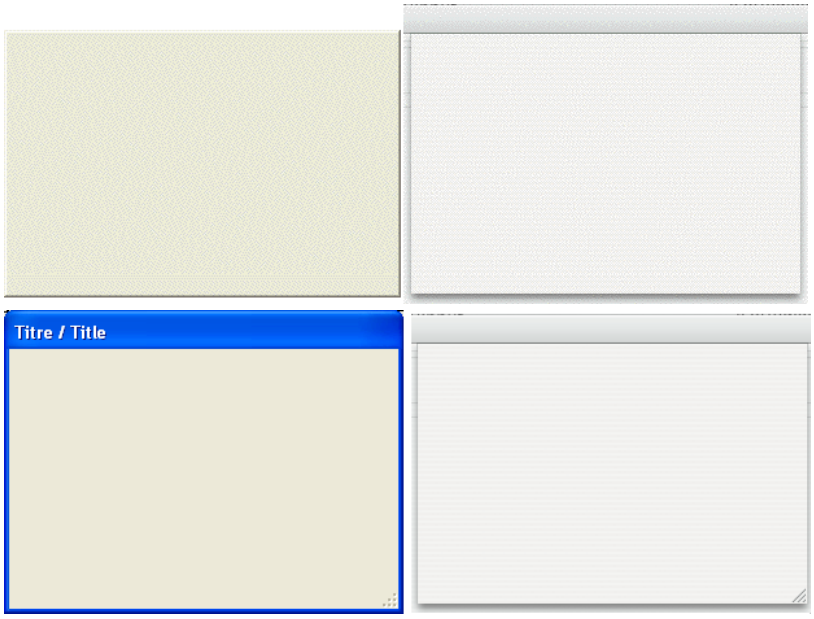
Note : Avec ce type de fenêtre, l'ensemble (constante + option) doit toujours être passé en valeur négative. Attention à passer par exemple -(Fenêtre palette+6) et non (-Fenêtre palette+6)

Fenêtre à coins arrondis (16)



- Peut avoir un titre : Oui
- Peut comporter une case de fermeture ou un équivalent : Oui
- Peut être redimensionnée : Non sous Mac OS
- Peut être agrandie/réduite ou "zoomée" : Non
- Adaptée aux barres de défilement : Non
- Utilisation : rare (obsolète)

Fenêtre feuille (33) et Fenêtre feuille redim (34)



Les fenêtres feuilles (sheet windows) sont des fenêtres spécifiques de l'interface Mac OS X. Ces fenêtres "descendent" de la barre de titre de la fenêtre principale via une animation et s'affichent par-dessus celle-ci. Elles sont automatiquement centrées dans la fenêtre principale. Leurs propriétés sont comparables à celles des boîtes de dialogue modales. Elles sont généralement utilisées pour effectuer une action en relation directe avec celle se déroulant dans la fenêtre principale.

- Il n'est possible de créer une fenêtre feuille sous Mac OS X que si la dernière fenêtre ouverte est visible et de type document (formulaire).
- La commande crée une fenêtre de type 1 (Dialogue modal) au lieu du type 33 et de type 8 (Fenêtre standard) au lieu du type 34 :
  - si la dernière fenêtre ouverte n'est pas visible ou n'est pas de type document,
  - sous Windows.
- Comme une fenêtre feuille doit être dessinée par-dessus un formulaire, son affichage est repoussé dans l'événement Sur chargement du premier formulaire chargé dans la fenêtre (cf. exemple 4 de la commande **Creer fenetre**).
- Utilisation : **DIALOGUE**, **AJOUTER ENREGISTREMENT**(...;...*) ou équivalent, sous Mac OS (non standard sous Windows).

Fenêtre pop up (32)

Ce type de fenêtre reprend les caractéristiques essentielles du type Dialogue simple (2) et dispose de propriétés avancées spécifiques :

- La fenêtre est automatiquement refermée avec annulation lorsque :
  - un clic se produit en-dehors de la fenêtre ;
  - la fenêtre d'arrière-plan ou la fenêtre MDI est déplacée ;
  - l'utilisateur appuie sur la touche **Echap** (ou **Esc**).
- Cette fenêtre s'affiche devant une fenêtre "parente" (elle ne doit d'ailleurs pas être utilisée comme fenêtre principale d'un process). La fenêtre d'arrière-plan n'est pas désactivée. En revanche, elle ne reçoit plus d'événement.
- Il n'est pas possible de redimensionner ou de déplacer la fenêtre à l'aide de la souris ; toutefois, lorsque cette opération est effectuée par programmation, le redessinement des éléments d'arrière-plan est optimisé.
- Utilisation : ce type de fenêtre est particulièrement adapté à la prise en charge des pop up menus associés aux boutons 3D de type "bevel" ou "barres outils".

- Limitations :
  - Il n'est pas possible d'afficher un objet pop up menu à l'intérieur d'une fenêtre de ce type.
  - A compter de la version 13 de 4D, ce type de fenêtre ne permet pas l'affichage des infobulles sous Mac OS.

Aspect texture (2048)

Sous Mac OS, il est possible d'appliquer l'apparence "texturée" aux fenêtres. Ce type d'apparence est largement répandu dans l'interface Macintosh. Sous Windows, cette propriété est sans effet.

Pour appliquer l'apparence "texture" à une fenêtre générée par la commande **Creer fenetre**, il suffit d'ajouter la constante Aspect texture au type de fenêtre défini dans le paramètre *type*. Par exemple :

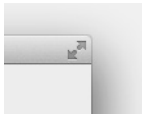
```
$fen:=Creer fenetre(10;80;-1;-1;Fenêtre standard+Aspect texture;"")
```

Cette apparence peut être associée aux types de fenêtres suivants :

Fenêtre standard
Fenêtre standard sans zoom
Fenêtre standard de taille fixe
Dialogue modal déplaçable
Fenêtre à coins arrondis

Avec mode plein écran Mac (65536)

L'option "plein écran" est disponible à compter de 4D v14 sous OS X pour les fenêtres de type document. Lorsque cette option est utilisée, le bouton "Plein écran" est affiché dans l'angle supérieur droit de la fenêtre :



Lorsque l'utilisateur clique sur cette icône, la fenêtre passe en plein écran et 4D masque automatiquement la barre d'outils principale.
Pour utiliser cette option, il suffit d'ajouter la constante Avec mode plein écran Mac au paramètre *type* pour les commandes **Creer fenetre**, **Creer fenetre formulaire** et **_o_Creer fenetre externe**. Par exemple, ce code crée une fenêtre formulaire avec bouton plein écran sous OS X :

```
$fen :=Creer fenetre formulaire([Interface];"Choix_User";Form fenêtre standard+Form avec mode plein écran Mac)
DIALOGUE([Interface];"Choix_User")
```

Note : Sous Windows, cette option n'a pas d'effet.

Fonctions mathématiques

- Abs
- Arctan
- Arrondi
- Convertisseur Euro
- Cos
- Dec
- Ent
- Exp
- FIXER NIVEAU COMPARAISON REEL
- Hasard
- Log
- Modulo
- Racine carree
- Sin
- Tan
- Troncature

Abs

Abs (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	⇒	Nombre dont vous voulez obtenir la valeur absolue
Résultat	Réel	⇐	Valeur absolue de nombre

Description

Abs retourne la valeur absolue (positive et sans signe) de *nombre*. Si *nombre* est négatif, sa valeur positive est retournée. Si *nombre* est positif, il est retourné inchangé.

Exemple

L'exemple suivant retourne la valeur absolue de -10,3, qui est 10,3 :

```
vVector:=Abs(-10,3)
```

Arctan

Arctan (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	→	Tangente pour laquelle vous souhaitez calculer l'angle en radians
Résultat	Réel	↩	Angle en radians

Description

Arctan retourne en radians la valeur de l'angle dont la tangente est spécifiée par *nombre*.

Note : 4D fournit les constantes prédéfinies Pi, Degré et Radian. Pi retourne le nombre Pi (3,14159...), Degré retourne la valeur en radians d'un degré (0,01745...) et Radian retourne la valeur en degrés d'un radian (57,29577...).

Exemple

Cet exemple permet d'afficher la valeur de Pi :

```
ALERTE("Pi est égal à : "+Chaine(Arctan(1)*4))
```

Arrondi

Arrondi (arrondi ; nbDécimales) -> Résultat

Paramètre	Type		Description
arrondi	Réel	⇒	Nombre à arrondir
nbDécimales	Entier long	⇒	Nombre de décimales de l'arrondi
Résultat	Réel	⇐	Valeur de nombre arrondie avec une précision égale à nbDécimales

Description

Arrondi retourne la valeur arrondie de *nombre* avec une précision égale à *nbDécimales*.

Si *nbDécimales* est positif, l'arrondi se fait sur la partie décimale de *nombre*. Si *nbDécimales* est négatif, l'arrondi se fait sur la partie entière de *nombre*.

Si le chiffre placé derrière le nombre de décimales défini par *nbDécimales* est compris entre 5 et 9, *nombre* est arrondi à la valeur supérieure s'il est positif et inférieure s'il est négatif. Si le chiffre placé derrière la dernière décimale est compris entre 0 et 4, la fonction arrondit *nombre* vers zéro.

Exemple

L'exemple suivant illustre la manière dont **Arrondi** fonctionne dans différents cas. A chaque ligne, une valeur est assignée à la variable *vRésultat*. Les commentaires décrivent le résultat :

```
vRésultat:=Arrondi(16,857;2) ` vRésultat vaut 16,86
vRésultat:=Arrondi(32345,67;-3) ` vRésultat vaut 32000
vRésultat:=Arrondi(29,8725;3) ` vRésultat vaut 29,873
vRésultat:=Arrondi(-1,5;0) ` vRésultat vaut -2
```

Convertisseur Euro

Convertisseur Euro (valeur ; deMonnaie ; versMonnaie) -> Résultat

Paramètre	Type		Description
valeur	Réel	⇒	Valeur à convertir
deMonnaie	Chaîne	⇒	Code ISO de la monnaie dans laquelle la valeur est exprimée
versMonnaie	Chaîne	⇒	Code ISO de la monnaie dans laquelle la valeur doit être convertie
Résultat	Réel	⇐	Valeur convertie

Description

La commande **Convertisseur Euro** vous permet d'effectuer tout type de conversion de valeurs entre les différentes monnaies des pays de la "zone euro" et l'Euro lui-même.

Vous pouvez convertir :

- une monnaie nationale en Euros,
- des Euros en une monnaie nationale,
- une monnaie nationale en une autre monnaie nationale. Dans ce cas, la conversion s'effectue toujours par l'intermédiaire de l'Euro, comme le stipule la réglementation. Par exemple, pour convertir des Francs belges en Marks allemands, 4D effectuera les conversions suivantes : Francs belges -> Euro -> Marks allemands.

Vous passez dans le premier paramètre la valeur à convertir.

Dans le second paramètre, vous indiquez le code ISO de la monnaie dans laquelle valeur est exprimée.

Dans le troisième paramètre, vous indiquez le code ISO de la monnaie dans laquelle vous souhaitez que valeur soit convertie.

Pour désigner les codes ISO, 4D vous propose les constantes prédéfinies suivantes, placées dans le thème "**Euro monnaies**" :

Constante	Valeur
Drachme grecque	GRD
Escudo portugais	PTE
Euro	EUR
Florin néerlandais	NLG
Franc belge	BEF
Franc français	FRF
Franc luxembourgeois	LUF
Lire italienne	ITL
Livre irlandaise	IEP
Mark allemand	DEM
Mark finlandais	FIM
Peseta espagnole	ESP
Schilling autrichien	ATS

Constante
Drachme grecque
Escudo portugais
Euro
Florin néerlandais
Franc belge
Franc français
Franc luxembourgeois
Lire italienne
Livre irlandaise
Mark allemand
Mark finlandais
Peseta espagnole
Schilling autrichien

Constante
Drachme grecque
Escudo portugais
Euro
Florin néerlandais
Franc belge
Franc français
Franc luxembourgeois
Lire italienne
Livre irlandaise
Mark allemand
Mark finlandais
Peseta espagnole
Schilling autrichien

Si nécessaire, 4D arrondit automatiquement le résultat des conversions et conserve 2 décimales — à l'exception des conversions vers les Lires italiennes, Francs luxembourgeois, Francs belges et Pesetas espagnoles, pour lesquelles 4D conserve 0 décimale (le résultat est un nombre entier).

La parité des différentes monnaies vis-à-vis de l'Euro est fixe. Les taux de conversion, utilisés en interne par 4D, sont les suivants :

Monnaie	Valeur pour 1 Euro
Drachme grecque	340,750
Escudo portugais	200,482
Florin néerlandais	2,20371
Franc belge	40,3399
Franc français	6,55957
Franc luxembourgeois	40,3399
Lire italienne	1936,27
Livre irlandaise	0,787564
Mark allemand	1,95583
Mark finlandais	5,94573
Peseta espagnole	166,386
Schilling autrichien	13,7603

Exemple

Voici différents types de conversion pouvant être obtenus à l'aide de cette commande :

```
$valeur:=10000 `Valeur exprimée en francs français
 `Convertir la valeur en euros
$EnEuros:=Convertisseur Euro($valeur;Franc français;Euro)
 `Convertir la valeur en lires italiennes
$EnLires:=Convertisseur Euro($valeur;Franc français;Lire italienne)
```

Cos

Cos (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	⇒	Nombre, exprimé en radians, dont vous voulez connaître le cosinus
Résultat	Réel	⇐	Cosinus de nombre

Description

Cos retourne le cosinus de *nombre*. La valeur *nombre* est exprimée en radians.

Note : 4D fournit les constantes prédéfinies Pi, Degré et Radian. Pi retourne le nombre Pi (3,14159...), Degré retourne la valeur en radians d'un degré (0,01745...) et Radian retourne la valeur en degrés d'un radian (57,29577...).

Dec

Dec (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	⇨	Valeur dont voulez obtenir la partie décimale
Résultat	Réel	⇦	Partie décimale de nombre

Description

Dec retourne la partie décimale de *nombre*. La valeur retournée est toujours positive ou nulle.

Exemple

L'exemple suivant utilise une valeur monétaire exprimée sous forme numérique et en extrait les parties "euros" et "centimes". Si *vrMontant* valait 7,31, *vlEuros* vaudrait 7 et *vlCentimes* 31 :

```
vlEuros:=Ent(vrMontant) ` Extraire les euros
vlCentimes:=Dec(vrMontant)*100 ` Extraire la partie décimale et la multiplier par 100 pour obtenir un entier
```

Ent

Ent (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	➡	Valeur dont vous voulez obtenir la partie entière
Résultat	Réel	➲	Partie entière de nombre

Description

Ent retourne la partie entière de *nombre* en l'arrondissant à l'entier inférieur.

Exemple

L'exemple suivant illustre le fonctionnement de **Ent** pour les nombres positifs et négatifs. Notez que la partie décimale du nombre est supprimée :

```
x:=Ent(123,4) ` x vaut 123
y:=Ent(-123,4) ` y vaut -124
```

Exp

Exp (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	⇒	Nombre à évaluer
Résultat	Réel	⇐	Exponentielle de nombre

Description

Exp retourne l'exponentielle (e=2,71828...) de *nombre*. **Exp** est la fonction inverse de **Log**.

Note : La fonction exponentielle, qui au nombre réel x fait correspondre le nombre réel y, est notée y = ex. 4D fournit la constante prédéfinie Nombre e (2,71828...).

Exemple

L'exemple suivant assigne l'exponentielle de 1 à *vrE* (le logarithme de *vrE* est 1) :

```
vrE:=Exp(1) ` vrE vaut e (e = 2,71828...)
```

FIXER NIVEAU COMPARAISON REEL

FIXER NIVEAU COMPARAISON REEL (epsilon)

Paramètre	Type		Description
epsilon	Réel	⇨	Valeur epsilon pour les comparaisons d'égalité des réels

Description

La commande **FIXER NIVEAU COMPARAISON REEL** définit la valeur *epsilon* utilisée par 4D lors d'une comparaison d'égalité des valeurs et expressions de type Réel.

Comme un ordinateur effectue des calculs approximatifs sur les réels, les tests sur l'égalité de valeurs réelles doivent tenir compte de cette approximation. Pour cela, 4D, lorsqu'il compare des valeurs réelles, teste en fait si la différence entre les deux valeurs est supérieure ou non à une certaine valeur. Cette valeur s'appelle l'**epsilon** et fonctionne de la manière suivante :
Soient deux valeurs réelles *a* et *b*. Si **Abs(a-b)** est supérieur à l'epsilon, les valeurs sont considérées comme différentes ; sinon, elles sont déclarées égales. Par défaut, 4D fixe la valeur epsilon à 10 à la puissance moins 6 (10^-6). Exemples :

- *0,00001=0,00002* retourne Faux car la différence *0,00001* est supérieure à *10^-6*.
- *0,000001=0,000002* retourne Vrai car la différence *0,000001* n'est pas supérieure à *10^-6*.
- *0,000001=0,000003* retourne Faux car la différence *0,000002* est supérieure à *10^-6*.

A l'aide de **FIXER NIVEAU COMPARAISON REEL**, vous pouvez augmenter ou réduire la valeur epsilon, en fonction de vos besoins.

Note : La commande n'aura pas d'effet si *epsilon* > 10^-3 ou si *epsilon* < 0.

Modifier l'epsilon affecte seulement la comparaison d'égalité des réels. Cela n'a pas d'effet sur les calculs et l'affichage des valeurs réelles.

ATTENTION : Cette commande doit être utilisée dans des cas spécifiques, comme par exemple un tri sur un champ dont les valeurs sont inférieures à 10^-6. En général, vous n'avez pas besoin de modifier la valeur par défaut d'epsilon.

Note : La commande **FIXER NIVEAU COMPARAISON REEL** n'a pas d'effet sur les recherches et les tris effectués avec les champs de type réel. Elle s'applique uniquement au langage de 4D.

Hasard

Hasard -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Nombre aléatoire

Description

Hasard retourne une valeur entière aléatoire comprise entre 0 et 32 767 (inclus).
Pour que la valeur aléatoire soit située dans un intervalle donné, utilisez la formule suivante :

```
(Hasard%(fin-début+1))+début
```

La valeur *début* est le premier nombre de l'intervalle, *fin* est le dernier.

Exemple

L'exemple suivant assigne une valeur entière aléatoire entre 10 et 30 à la variable *vRésultat* :

```
vRésultat:=(Hasard% 21)+10
```

Log

Log (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	⇒	Nombre dont vous voulez obtenir le logarithme népérien
Résultat	Réel	⇐	Logarithme népérien de nombre

Description

Log retourne le logarithme népérien de *nombre*. **Log** est la fonction inverse de **Exp**.
Note : 4D fournit la constante prédéfinie Nombre e (2,71828...).

Exemple

L'exemple suivant affiche 1 :

```
ALERTE(Chaine(Log(Exp(1))))
```

Modulo

Modulo (nombre1 ; nombre2) -> Résultat

Paramètre	Type		Description
nombre1	Entier long	⇒	Nombre à diviser (numérateur)
nombre2	Entier long	⇒	Nombre diviseur (dénominateur)
Résultat	Réel	⇐	Reste de la division entière de nombre1 par nombre2

Description

La fonction **Modulo** divise *nombre1* par *nombre2* et retourne le reste sous forme d'un nombre entier.

Notes :

- **Modulo** accepte des expressions de type Entier, Entier long et Réel (numérique). Cependant, si *nombre1* et/ou *nombre2* sont des nombres réels, ils sont arrondis avant le calcul du **Modulo**.
- La fonction **Modulo** est à utiliser avec précaution avec des nombres réels de grande taille (au-delà de 2^31). Dans ce cas en effet, son fonctionnement peut se heurter aux limites des capacités de calcul des processeurs standard.

Vous pouvez également utiliser l'opérateur "%" pour calculer le reste d'une division (reportez-vous à la section **Opérateurs numériques**). Toutefois, cet opérateur retourne des résultats valides uniquement avec des expressions de type Entier et Entier long. Si vous voulez calculer le modulo de nombres réels, vous devez utiliser la commande **Modulo**.

Exemple

L'exemple suivant illustre le fonctionnement de **Modulo** dans différents cas de figure. A chaque ligne, un nombre est assigné à la variable *vRésultat*. Les commentaires fournissent le résultat obtenu :

```
vRésultat:=Modulo(3;2) ` vRésultat prend la valeur 1
vRésultat:=Modulo(4;2) ` vRésultat prend la valeur 0
vRésultat:=Modulo(3,5;2) ` vRésultat prend la valeur 0
```

Racine carree

Racine carree (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	⇒	Nombre duquel calculer la racine carrée
Résultat	Réel	⇐	Racine carrée de nombre

Description

Racine carree retourne la racine carrée de *nombre*.

Exemple 1

La ligne :

```
$vrRacineDeDeux :=Racine carree(2)
```

affecte la valeur *1,414213562373* à la variable *$vrRacineDeDeux*.

Exemple 2

La méthode listée ci-dessous retourne l'hypoténuse du triangle rectangle dont les deux côtés sont passés en paramètres :

```
  ` Méthode Hypoténuse
  ` Hypoténuse ( Réel ; Réel ) -> Réel
  ` Hypoténuse ( côtéA ; côtéB ) -> Hypoténuse
C_REEL($0;$1;$2)
$0:=Racine carree(($1^2)+($2^2))
```

Par exemple, ***Hypoténuse (4;3)*** retourne 5.

Sin

Sin (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	⇒	Nombre, exprimé en radians, dont vous voulez connaître le sinus
Résultat	Réel	⇐	Sinus de nombre

Description

Sin retourne le sinus de *nombre*. La valeur *nombre* est exprimée en radians.

Note : 4D fournit les constantes prédéfinies Pi, Degré et Radian. Pi retourne le nombre Pi (3,14159...), Degré retourne la valeur en radians d'un degré (0,01745...) et Radian retourne la valeur en degrés d'un radian (57,29577...).

Tan

Tan (nombre) -> Résultat

Paramètre	Type		Description
nombre	Réel	⇒	Nombre, exprimé en radians, dont vous voulez connaître la tangente
Résultat	Réel	⇐	Tangente de nombre

Description

Tan retourne la tangente de *nombre*. La valeur *nombre* est exprimée en radians.

Note : 4D fournit les constantes prédéfinies Pi, Degré et Radian. Pi retourne le nombre Pi (3,14159...), Degré retourne la valeur en radians d'un degré (0,01745...) et Radian retourne la valeur en degrés d'un radian (57,29577...).

Troncature

Troncature (nombre ; nbDécimales) -> Résultat

Paramètre	Type		Description
nombre	Réel	⇒	Nombre à tronquer
nbDécimales	Entier long	⇒	Nombre de décimales à conserver
Résultat	Réel	⇐	nombre tronqué à partir du nombre de décimales indiqué par nbDécimales

Description

Troncature retourne *nombre* dont la partie décimale a été tronquée à partir du nombre de décimales spécifié par *nbDécimales*. **Troncature** arrondit toujours *nombre* à la valeur inférieure.

Si *nbDécimales* est positif, la troncature se fait sur la partie décimale de *nombre*. Si *nbDécimales* est négatif, la troncature se fait sur la partie entière de *nombre*.

Exemple

L'exemple suivant illustre la manière dont **Troncature** fonctionne dans différents cas. A chaque ligne, une valeur est assignée à la variable *vRésultat*. Les commentaires décrivent le résultat :

```
vRésultat:=Troncature(216,897;1) ` Résultat prend la valeur 216,8
vRésultat:=Troncature(216,897;-1) ` Résultat prend la valeur 210
vRésultat:=Troncature(-216,897;1) ` Résultat prend la valeur -216,9
vRésultat:=Troncature(-216,897;-1) ` Résultat prend la valeur -220
```

Fonctions statistiques

Présentation des fonctions statistiques

Les fonctions de ce thème effectuent des calculs sur des séries de valeurs.

Les fonctions **Moyenne**, **Min**, **Max**, **Somme**, **Somme des carres**, **Ecart type** et **Variance** s'appliquent soit à des champs, soit à des tableaux :

- lorsqu'elles s'appliquent à des champs, elles utilisent la sélection courante d'enregistrements,
- lorsqu'elles s'appliquent à des tableaux, elles utilisent les éléments du tableau.

A noter que lorsqu'elles s'appliquent à des champs, les fonctions **Somme des carres**, **Ecart type** et **Variance** ne peuvent être utilisées que pendant l'impression.

Toutes ces fonctions ne travaillent que sur des valeurs numériques et retournent des valeurs numériques.

Utilisation des fonctions statistiques hors impression

Lorsque les fonctions **Moyenne**, **Min**, **Max** et **Somme** sont utilisées sur un champ en-dehors d'une opération d'impression, il se peut qu'elles aient à charger chaque enregistrement de la sélection courante pour calculer le résultat. S'il y a beaucoup d'enregistrements, l'opération peut être longue. Pour limiter la durée du traitement, vous pouvez indexer le champ.

Note : Lorsque l'opération est longue, un thermomètre de progression apparaît. Ce thermomètre comporte un bouton Arrêt permettant à l'utilisateur d'interrompre l'opération. Si l'utilisateur clique sur ce bouton, la variable OK prend la valeur 0. Si l'opération est correctemen menée à son terme, la variable OK prend la valeur 1.

Utilisation des fonctions statistiques dans un état imprimé

Lorsque les fonctions statistiques sont utilisées dans un état, elles se comportent de façon particulière, car c'est l'état lui-même qui charge chaque enregistrement. Utilisez ces fonctions dans une méthode formulaire ou objet quand vous imprimez avec la commande **IMPRIMER SELECTION** ou quand vous imprimez en mode Développement à l'aide de la commande **Imprimer** dans le menu **Fichier**.

Lorsque vous utilisez ces fonctions dans un état, les valeurs retournées sont fiables uniquement dans le niveau 0 de rupture et seulement si la phase de traitement des ruptures est active. Cela signifie qu'elles ne sont utiles qu'en fin d'état, lorsque tous les enregistrements ont été traités.

Vous ne devez utiliser ces fonctions avec une méthode objet que dans une zone non-saisissable incluse dans la zone de rupture R0.

Pour mémoire : un champ passé comme paramètre à une fonction statistique doit être un numérique.

Ecart type

Ecart type (séries) -> Résultat

Paramètre	Type		Description
séries	Champ, Tableau	⇒	Valeurs dont vous voulez obtenir l'écart type
Résultat	Réel	⇐	Ecart type de séries

Description

Ecart type retourne l'écart type (c.-à-d. la racine carrée de la variance) de *séries*.

Si *séries* est un champ indexé, l'index sera utilisé pour le calcul.

Vous pouvez passer dans *séries* un tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

Exemple 1

L'exemple suivant est la méthode objet d'une variable appelée **EcartT**. La méthode assigne l'écart type d'une série à **EcartT** :

```
EcartT:=Ecart type([Table1]SérieValeurs)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection et activer la phase de rupture :

```
TOUT SELECTIONNER([Table1])
TRIER([Table1];[Table1]SérieValeurs;>)
NIVEAUX DE RUPTURES(1)
CUMULER SUR([Table1]SérieValeurs)
FORMULAIRE SORTIE([Table1];"FormImpression")
IMPRIMER SELECTION([Table1])
```

Note : La valeur du paramètre de la commande **NIVEAUX DE RUPTURES** doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème **Impressions**.

Exemple 2

Cet exemple vous permet d'obtenir l'écart type d'une série de valeurs placées dans un tableau :

```
TABLEAU REEL($TabNote;0)
CHERCHER([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION VERS TABLEAU([Exams]Exam_Note;$TabNote)
vEcartT:=Ecart type($TabNote)
```

Max

Max (séries {; cheminAttribut}) -> Résultat

Paramètre	Type		Description
séries	Champ, Tableau	➡	Valeurs desquelles dont vous voulez obtenir la plus élevée
cheminAttribut	Texte	➡	Chemin d'attribut duquel calculer la valeur maximale
Résultat	Réel	⬅	Valeur la plus élevée de séries

Description

Max retourne la valeur la plus élevée contenue dans *séries*.

Si *séries* est un champ indexé, l'index est utilisé pour la recherche de cette valeur.

Vous pouvez passer dans *séries* un tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

Si la sélection de *séries* est vide, **Max** retourne 0.

La commande accepte un paramètre optionnel de type texte, *cheminAttribut*, que vous pouvez utiliser si *séries* est un champ de type Objet. Il vous permet de définir le chemin de l'attribut dont le contenu doit être utilisé pour le calcul de la valeur maximale. Utilisez la notation à points standard pour définir le chemin dans l'objet jusqu'à l'attribut, par exemple "Clients.enfants.age". Attention, gardez à l'esprit que les noms d'attributs d'objets tiennent compte de la casse des caractères.
Seules les valeurs numériques des attributs sont utilisées pour le calcul. Si l'attribut contient des valeurs non numériques, elles sont ignorées.

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Si elle est interrompue (par exemple si l'utilisateur clique sur le bouton **Arrêt** dans le thermomètre de progression), la variable OK prend la valeur 0.

Exemple 1

L'exemple ci-dessous est la méthode objet d'une variable, *vMax*, placée dans la zone de rupture R0 d'un formulaire sortie. La variable est imprimée à la fin de l'état. La méthode objet assigne la valeur la plus élevée du champ à la variable, qui est alors imprimée dans la dernière rupture de l'état.

```
vMax:=Max([Employés]Salaire)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection courante et activer la phase de rupture :

```
TOUT SELECTIONNER([Employés])
TRIER([Employés];[Employés]Nom;>)
NIVEAUX DE RUPTURES(1)
CUMULER SUR([Employés]Salaire)
FORM FIXER SORTIE([Employés];"FormImpression")
IMPRIMER SELECTION([Employés])
```

Note : La valeur du paramètre de la commande **NIVEAUX DE RUPTURES** doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème **Impressions**.

Exemple 2

Cet exemple vous permet d'obtenir la valeur la plus élevée d'un tableau :

```
TABLEAU REEL($TabNote;0)
CHERCHER([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION VERS TABLEAU([Exams]Exam_Note;$TabNote)
vMax:=Max($TabNote)
```

Exemple 3

Pour un exemple de calcul utilisant un attribut de champ objet, veuillez vous reporter à l'exemple 3 de la commande **Moyenne**.

Min

Min (séries {; cheminAttribut}) -> Résultat

Paramètre	Type		Description
séries	Champ, Tableau	⇒	Valeurs desquelles vous voulez obtenir la plus basse
cheminAttribut	Texte	⇒	Chemin d'attribut duquel calculer la valeur minimale
Résultat	Réel	⇐	Valeur la plus basse de séries

Description

Min retourne la valeur la plus faible de *séries*. Si *séries* est un champ indexé, l'index est utilisé pour la recherche de cette valeur.

Vous pouvez passer dans *séries* un tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

Si la sélection de *séries* est vide, **Min** retourne 0.

La commande accepte un paramètre optionnel de type texte, *cheminAttribut*, que vous pouvez utiliser si *séries* est un champ de type Objet. Il vous permet de définir le chemin de l'attribut dont le contenu doit être utilisé pour le calcul de la valeur minimale. Utilisez la notation à points standard pour définir le chemin dans l'objet jusqu'à l'attribut, par exemple "Clients.enfants.age". Attention, gardez à l'esprit que les noms d'attributs d'objets tiennent compte de la casse des caractères.
Seules les valeurs numériques des attributs sont utilisées pour le calcul. Si l'attribut contient des valeurs non numériques, elles sont ignorées.

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Si elle est interrompue (par exemple si l'utilisateur clique sur le bouton **Arrêt** dans le thermomètre de progression), la variable OK prend la valeur 0.

Exemple 1

L'exemple ci-dessous est la méthode objet d'une variable, *vMin*, placée dans la zone de rupture R0 d'un formulaire sortie. La variable est imprimée à la fin de l'état. La méthode objet assigne la valeur la plus basse du champ à la variable, qui est alors imprimée dans la dernière rupture de l'état.

```
vMin:=Min([Employés] Salaire)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection courante et activer la phase de rupture :

```
TOUT SELECTIONNER([Employés])
TRIER([Employés];[Employés]Nom;>)
NIVEAUX DE RUPTURES(1)
CUMULER SUR([Employés]Salaire)
FORM FIXER SORTIE([Employés];"FormImpression")
IMPRIMER SELECTION([Employés])
```

Note : La valeur du paramètre de la commande **NIVEAUX DE RUPTURES** doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème **Impressions**.

Exemple 2

L'exemple suivant recherche la plus petite vente d'un employé et affiche le résultat dans une boîte de dialogue d'alerte :

```
ALERTE("Plus petite vente = "+Chaine(Min([Employés]Ventes)))
```

Exemple 3

Cet exemple vous permet d'obtenir la valeur la plus basse d'un tableau :

```
TABLEAU REEL($TabNote;0)
CHERCHER([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION VERS TABLEAU([Exams]Exam_Note;$TabNote)
vMin:=Min($TabNote)
```

Exemple 4

Pour un exemple de calcul utilisant un attribut de champ objet, veuillez vous reporter à l'exemple 3 de la commande **Moyenne**.

Moyenne

Moyenne (séries {; cheminAttribut}) -> Résultat

Paramètre	Type		Description
séries	Champ, Tableau	➡	Valeurs dont vous voulez calculer la moyenne
cheminAttribut	Texte	➡	Chemin d'attribut duquel calculer la moyenne
Résultat	Réel	⬅	Moyenne arithmétique de séries

Description

Moyenne retourne la moyenne arithmétique de *séries*. Si *séries* est un champ indexé, l'index est utilisé pour le calcul.

Vous pouvez passer dans *séries* un tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

La commande accepte un paramètre optionnel de type texte, *cheminAttribut*, que vous pouvez utiliser si *séries* est un champ de type Objet. Il vous permet de définir le chemin de l'attribut dont le contenu doit être utilisé pour le calcul de la moyenne. Utilisez la notation à points standard pour définir le chemin dans l'objet jusqu'à l'attribut, par exemple "Clients.enfants.age". Attention, gardez à l'esprit que les noms d'attributs d'objets tiennent compte de la casse des caractères.
Seules les valeurs numériques des attributs sont utilisées pour le calcul. Si l'attribut contient des valeurs non numériques, elles sont ignorées.

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Si elle est interrompue (par exemple si l'utilisateur clique sur le bouton **Arrêt** dans le thermomètre de progression), la variable OK prend la valeur 0.

Exemple 1

Dans l'exemple suivant, une valeur est assignée à une variable se trouvant dans la zone de rupture R0 d'un formulaire sortie. La ligne de code ci-dessous constitue la méthode objet de la variable. Elle n'est exécutée qu'à l'impression du niveau de rupture 0 :

```
vMoyenne:=Moyenne([Employés]Salaire)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection courante et activer la phase de rupture :

```
TOUT SELECTIONNER([Employés])
TRIER([Employés];[Employés]Nom;>)
NIVEAUX DE RUPTURES(1)
CUMULER SUR([Employés]Salaire)
FORM FIXER SORTIE([Employés];"FormImpression")
IMPRIMER SELECTION([Employés])
```

Note : La valeur du paramètre de la commande **NIVEAUX DE RUPTURES** doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème **Impressions**.

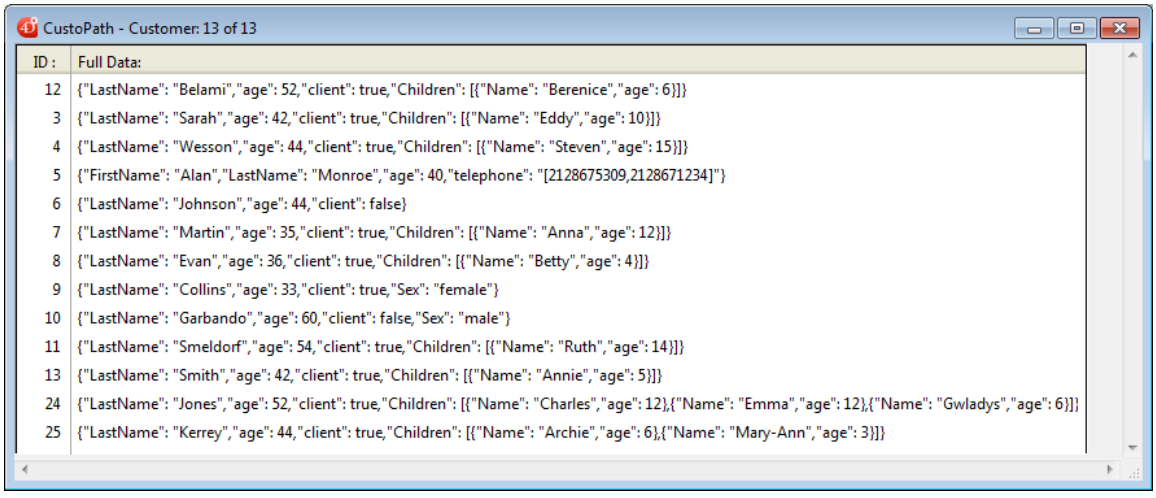
Exemple 2

Cet exemple vous permet d'obtenir la moyenne des 15 premières notes de la sélection :

```
TABLEAU REEL($TabNote;0)
CHERCHER([Exams];[Exams]Exam_Date=!01/07/11!)
TRIER([Exams];[Exams]Exam_Note;<)
SELECTION VERS TABLEAU([Exams]Exam_Note;$TabNote)
TABLEAU REEL($TabNote;15)
vAverage:=Moyenne($TabNote)
```

Exemple 3

Votre table [Customer] comporte un champ objet "full_Data" contenant les données suivantes :



Vous pouvez effectuer les calculs suivants :

```
C_REEL($vAvg)
TOUT SELECTIONNER([Customer])
$vAvg:=Moyenne([Customer]full_Data;"age")
 //$vAvg vaut 44,46

C_ENTIER LONG($vTot)
$vTot:=Somme([Customer]full_Data;"Children[].age")
 //$vTot vaut 105
```

Somme

Somme (séries {; cheminAttribut}) -> Résultat

Paramètre	Type		Description
séries	Champ, Tableau	➡	Valeurs dont vous souhaitez calculer la somme
cheminAttribut	Texte	➡	Chemin d'attribut duquel calculer la somme
Résultat	Réel	⮌	Somme de séries

Description

Somme retourne la somme (c'est-à-dire le total de toutes les valeurs) de *séries*. Si *séries* est un champ indexé, l'index est utilisé pour le calcul.

Vous pouvez passer dans *séries* un tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

La commande accepte un paramètre optionnel de type texte, *cheminAttribut*, que vous pouvez utiliser si *séries* est un champ de type Objet. Il vous permet de définir le chemin de l'attribut dont le contenu doit être utilisé pour le calcul de la somme des valeurs. Utilisez la notation à points standard pour définir le chemin dans l'objet jusqu'à l'attribut, par exemple "Clients.enfants.age". Attention, gardez à l'esprit que les noms d'attributs d'objets tiennent compte de la casse des caractères.
Seules les valeurs numériques des attributs sont utilisées pour le calcul. Si l'attribut contient des valeurs non numériques, elles sont ignorées.

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Si elle est interrompue (par exemple si l'utilisateur clique sur le bouton **Arrêt** dans le thermomètre de progression), la variable OK prend la valeur 0.

Exemple 1

L'exemple ci-dessous est la méthode objet d'une variable, *vTotal*, placée dans un formulaire. La méthode assigne à la variable la somme de tous les salaires :

```
vTotal:=Somme([Employés]Salaire)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection courante et activer le traitement des ruptures :

```
TOUT SELECTIONNER([Employés])
TRIER([Employés];[Employés]Nom;>)
NIVEAUX DE RUPTURES(1)
CUMULER SUR([Employés]Salaire)
FORMULAIRE SORTIE([Employés];"FormImpression")
IMPRIMER SELECTION([Employés])
```

Note : La valeur du paramètre de la commande **NIVEAUX DE RUPTURES** doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème **Impressions**.

Exemple 2

Cet exemple vous permet d'obtenir la somme de toutes les valeurs placées dans un tableau :

```
TABLEAU REEL($TabNote;0)
CHERCHER([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION VERS TABLEAU([Exams]Exam_Note;$TabNote)
vSomme:=Somme($TabNote)
```

Exemple 3

Pour un exemple de calcul utilisant un attribut de champ objet, veuillez vous reporter à l'exemple 3 de la commande **Moyenne**.

Somme des carres

Somme des carres (séries) -> Résultat

Paramètre	Type		Description
séries	Champ, Tableau	⇒	Valeurs dont vous voulez obtenir la somme des carrés
Résultat	Réel	⇐	Somme des carrés de séries

Description

Somme des carres retourne la somme des carrés de *séries*. Si *séries* est un champ indexé, l'index est utilisé pour le calcul.

Vous pouvez passer dans *séries* un tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

Exemple 1

L'exemple suivant est la méthode objet d'une variable appelée Carrés. La méthode assigne la somme des carrés d'une série de valeurs à Carrés. La méthode est imprimée dans la dernière rupture de l'état :

```
Carrés:=Somme des carres([Table1]SérieValeurs)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection et activer la phase de rupture :

```
TOUT SELECTIONNER([Table1])
TRIER([Table1];[Table1]SérieValeurs;>)
NIVEAUX DE RUPTURES(1)
CUMULER SUR([Table1]SérieValeurs)
FORMULAIRE SORTIE([Table1];"FormImpression")
IMPRIMER SELECTION([Table1])
```

Note : La valeur du paramètre de la commande **NIVEAUX DE RUPTURES** doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème **Impressions**.

Exemple 2

Cet exemple vous permet d'obtenir la somme des carrés des valeurs placées dans un tableau :

```
TABLEAU REEL($TabNote;0)
CHERCHER([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION VERS TABLEAU([Exams]Exam_Note;$TabNote)
vSommeCarres:=Somme des carres($TabNote)
```

Variance

Variance (séries) -> Résultat

Paramètre	Type		Description
séries	Champ, Tableau	⇒	Valeurs dont vous voulez obtenir la variance
Résultat	Réel	⇐	Variance de séries

Description

Variance retourne la variance de *séries*. Si *séries* est un champ indexé, l'index est utilisé pour le calcul.

Vous pouvez passer dans *séries* un tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

La variance d'un ensemble de valeurs est la moyenne des carrés des écarts par rapport à la moyenne. C'est une valeur de dispersion autour de la moyenne. 4D utilise la formule de variance suivante :

Variance(x) = Somme (x-m)*(x-m)/(n-1)
m = Moyenne
n = Nombre de valeurs

Si les valeurs considérées ne constituent pas un échantillon, multipliez la valeur retournée par **Variance** par (n-1)/n.

Exemple 1

L'exemple suivant est la méthode objet d'une variable appelée **Var**. La méthode assigne la variance de la série à **Var**:

```
Var:=Variance([Etudiants]Notes)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection et activer la phase de rupture :

```
TOUT SELECTIONNER([Etudiants])
TRIER([Etudiants];[Etudiants]Classe;>)
NIVEAUX DE RUPTURES(1)
CUMULER SUR([Etudiants]Notes)
FORMULAIRE SORTIE([Etudiants];"FormImpression")
IMPRIMER SELECTION([Etudiants])
```

Note : La valeur du paramètre de la commande **NIVEAUX DE RUPTURES** doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème **Impressions**.

Exemple 2

Cet exemple vous permet d'obtenir la variance des valeurs placées dans un tableau :

```
TABLEAU REEL($TabNote;0)
CHERCHER([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION VERS TABLEAU([Exams]Exam_Note;$TabNote)
vVariance:=Variance($TabNote)
```

Formulaires

- APPELER FORMULAIRE
- FORM ALLER A PAGE
- FORM CAPTURE ECRAN
- FORM CHARGER
- FORM DERNIERE PAGE
- FORM FIXER ENTREE
- FORM FIXER REDIMENSIONNEMENT HORIZONTAL
- FORM FIXER REDIMENSIONNEMENT VERTICAL
- FORM FIXER SORTIE
- FORM FIXER TAILLE
- FORM LIBERER
- FORM LIRE OBJETS
- FORM Lire page courante
- FORM LIRE PARAMETRE
- FORM LIRE PROPRIETES
- FORM LIRE REDIMENSIONNEMENT HORIZONTAL
- FORM LIRE REDIMENSIONNEMENT VERTICAL
- FORM PAGE PRECEDENTE
- FORM PAGE SUIVANTE
- FORM PREMIERE PAGE
- Nom du formulaire courant

APPELER FORMULAIRE

APPELER FORMULAIRE (fenêtre ; méthode {; param}{; param2 ; ... ; paramN})

Paramètre	Type		Description
fenêtre	RefFen	➡	Numéro de référence de la fenêtre
méthode	Texte	➡	Nom de la méthode projet à appeler
param	Expression	➡	Paramètre(s) passé(s) à la méthode

Description

La commande **APPELER FORMULAIRE** exécute la méthode projet dont le nom est passé dans *méthode* avec un ou plusieurs *param*(s) optionnel(s) dans le contexte d'un formulaire affiché dans la *fenêtre*, indépendamment du process auquel appartient la fenêtre.

Tout comme dans la communication interprocess basée sur les workers (voir **A propos des workers**), une boîte aux lettres est associée à chaque fenêtre et peut être utilisée lorsque la fenêtre affiche un formulaire (après l'événement Sur chargement). **APPELER FORMULAIRE** encapsule le nom de la méthode et ses arguments dans un message envoyé dans la boîte aux lettres de la fenêtre. Le formulaire exécute alors le message dans son propre process. Le process appelant peut être coopératif ou préemptif, par conséquent cette fonctionnalité permet à un process préemptif d'échanger des informations avec des formulaires.

Dans *fenêtre*, passez le numéro de référence de la fenêtre affichant le formulaire appelé.

Dans *méthode*, passez le nom de la méthode projet qui doit être exécutée dans le contexte du process parent de la *fenêtre*.

Vous pouvez aussi passer des paramètres à la méthode en utilisant un ou plusieurs paramètres *param*. Vous passez les paramètres de la même façon que vous les passeriez à une sous-routine (voir la section **Passer des paramètres aux méthodes**). Lors de l'exécution dans le contexte du formulaire, la méthode reçoit les valeurs des paramètres dans *$1*, *$2*, et ainsi de suite. N'oubliez pas que des tableaux ne peuvent pas être passés en paramètre d'une méthode. En outre, dans le contexte de la commande **APPELER FORMULAIRE**, les points suivants doivent être pris en compte :

- Les pointeurs sur les tables et les champs sont autorisés.
- Les pointeurs sur les variables, en particulier les variables locales ou process, ne sont pas recommandés car ces variables peuvent être indéfinies au moment où la méthode du process tente d'y accéder.
- Si vous passez un paramètre de type Objet, 4D crée une copie de l'objet dans le process de destination si le formulaire est dans un process différent de celui appelant la commande **APPELER FORMULAIRE**.

Exemple 1

Vous voulez ouvrir deux fenêtres de dialogue à partir d'un même formulaire, mais avec des couleurs de fonds différentes et des messages différents. Vous souhaitez également envoyer des messages par la suite et les afficher dans chaque fenêtre de dialogue.

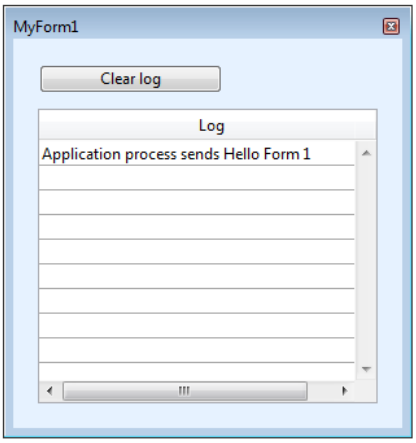
Dans le formulaire principal, un bouton ouvre les deux dialogues :

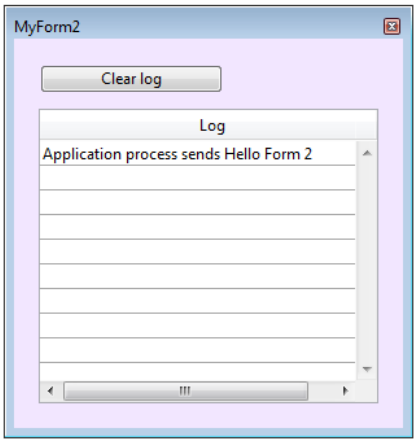
```
 //Méthode objet pour créer les formulaires
 //Première fenêtre
formRef1:=Creer fenetre formulaire("FormMessage";Form fenêtre palette;A gauche)
CHANGER TITRE FENETRE("MyForm1";formRef1)
DIALOGUE("FormMessage";*)
AFFICHER FENETRE(formRef1)

 //Seconde fenêtre
formRef2:=Creer fenetre formulaire("FormMessage";Form fenêtre palette;A gauche+500)
CHANGER TITRE FENETRE("MyForm2";formRef2)
DIALOGUE("FormMessage";*)
AFFICHER FENETRE(formRef2)

 //Envoi des couleurs
APPELER FORMULAIRE(formRef1;"doSetColor";0x00E6F2FF)
APPELER FORMULAIRE(formRef2;"doSetColor";0x00F2E6FF)
 //Création des messages
APPELER FORMULAIRE(formRef1;"doAddMessage";Nom du process courant;"Hello Form 1")
APPELER FORMULAIRE(formRef2;"doAddMessage";Nom du process courant;"Hello Form 2")
```

La méthode *doAddMessage* ajoute simplement une ligne dans la list box du formulaire "FormMessage" :

```
C_TEXTE($1) //Origine du message
C_TEXTE($2) //Message à afficher
 //Récupère le message contenu dans $2 et l'envoie dans la list box
$p:=OBJET Lire pointeur(Objet nommé;"Column1")
INSERER DANS TABLEAU($p->;1)
$p->{1}:=$1+" sends "+$2
```

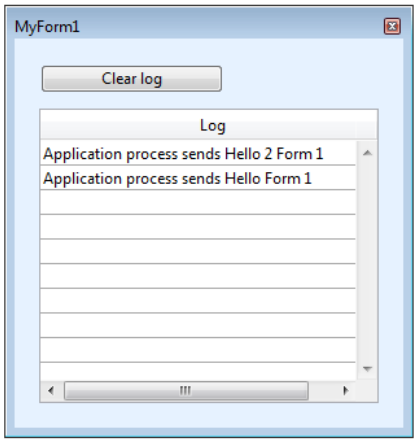
A l'utilisation, vous obtenez le résultat suivant :

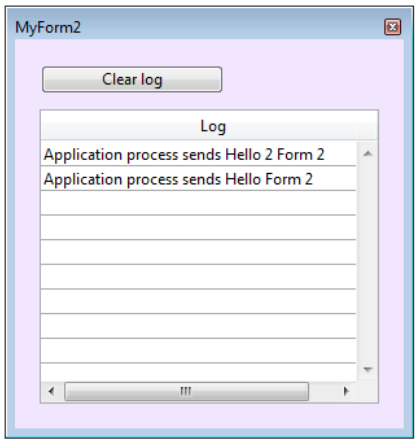




Vous pouvez alors ajouter d'autres messages en exécutant à nouveau la commande **APPELER FORMULAIRE** :

```
APPELER FORMULAIRE(formRef1;"doAddMessage";Nom du process courant;"Hello 2 Form 1")
APPELER FORMULAIRE(formRef2;"doAddMessage";Nom du process courant;"Hello 2 Form 2")
```





Exemple 2

Vous pouvez utiliser la commande **APPELER FORMULAIRE** pour passer des paramètres personnalisés à un formulaire, par exemple des valeurs de configuration, sans avoir à utiliser des variables process :

```
$win:=Creer fenetre formulaire("form")
APPELER FORMULAIRE($win;"configure";param1;param2)
DIALOGUE("form")
```

FORM ALLER A PAGE

FORM ALLER A PAGE (numéroPage {; *})

Paramètre	Type		Description
numéroPage	Entier long	⇒	Numéro de la page à afficher
*	Opérateur	⇒	Changer la page du sous-formulaire courant

Description

FORM ALLER A PAGE change la page courante du formulaire pour afficher la page désignée par *numéroPage*.

Si aucun formulaire n'est affiché ou chargé via la commande **FORM CHARGER**, ou si *numéroPage* correspond à la page courante du formulaire, **FORM ALLER A PAGE** ne fait rien. Si *numéroPage* est supérieur au nombre de pages du formulaire, la dernière page est affichée. Si *numéroPage* est inférieur à 1, la première page est affichée.

Le paramètre * est utile lorsque la commande est appelée dans le contexte d'un sous-formulaire en page contenant plusieurs pages. Dans ce cas, si vous passez ce paramètre, la commande change la page du sous-formulaire courant (celui qui a appelé la commande). Par défaut, si le paramètre * est omis, la commande s'applique toujours au formulaire parent.

A propos des commandes de gestion des pages

4D vous fournit des actions automatiques pour les boutons qui effectuent les mêmes tâches que les commandes **FORM PREMIERE PAGE**, **FORM DERNIERE PAGE**, **FORM PAGE SUIVANTE**, **FORM PAGE PRECEDENTE**, **FORM ALLER A PAGE** que vous pouvez associer aux objets tels que les onglets, les listes déroulantes, etc. A chaque fois que c'est possible, utilisez les actions automatiques pour les boutons plutôt que ces commandes.
Les commandes de gestion des pages peuvent être utilisées avec des formulaires entrée ou des formulaires affichés dans des boîtes de dialogue. Les formulaires sortie n'utilisent que la première page. Un formulaire comprend toujours au minimum une page, la première. Notez bien que quel que soit le nombre de pages qu'il contient, un formulaire ne peut être associé qu'à une seule méthode formulaire.

- Vous pouvez utiliser la commande **FORM Lire page courante** pour savoir quelle page est affichée à l'écran.
- Vous pouvez utiliser l'**Evenement formulaire** Sur changement page qui est généré à chaque fois que le formulaire change de page courante.

Note : Pendant que vous construisez un formulaire, vous pouvez utiliser les pages 1 à N du formulaire ainsi que la page 0 (zéro), dans laquelle vous placez les objets que vous voulez faire apparaître sur toutes les pages. Lors de l'utilisation du formulaire, et donc lorsque les commandes de gestion des pages sont appelées, seules les pages 1 à N sont accessibles : la page 0 est automatiquement combinée à la page affichée à l'écran.

Exemple

L'exemple suivant est la méthode objet d'un bouton affichant la page 3 du formulaire :

```
FORM ALLER A PAGE(3)
```

FORM CAPTURE ECRAN

FORM CAPTURE ECRAN ({{laTable ;} nomFormulaire ;} imageForm {; pageNum})

Paramètre	Type		Description
laTable	Table	⇒	Table du formulaire
nomFormulaire	Texte	⇒	Nom du formulaire
imageForm	Image	⇐	Image du formulaire en exécution si premier(s) paramètre(s) omis, ou Image du formulaire dans l'éditeur de formulaires si un nom de formulaire est passé
pageNum	Entier long	⇒	Numéro de page du formulaire

Description

La commande **FORM CAPTURE ECRAN** retourne un formulaire sous forme d'image. Cette commande admet deux syntaxes différentes : en fonction de la syntaxe utilisée, vous pouvez obtenir soit l'image d'un formulaire exécuté, soit l'image du formulaire dans l'éditeur de formulaires.

- FORM CAPTURE ECRAN (*imageForm*)
  Cette syntaxe vous permet d'obtenir une capture écran exacte de la page courante du formulaire en cours d'exécution ou chargé via la commande **FORM CHARGER** : l'image retournée dans le paramètre *imageForm* contient la totalité des objets visibles du formulaire avec les valeurs courantes des champs et des variables du formulaire, les sous-formulaires, etc. Le formulaire est retourné intégralement, sans tenir compte de la taille de la fenêtre qui le contient.
  A noter que cette syntaxe ne fonctionne qu'avec les formulaires d'entrée.

- FORM CAPTURE ECRAN ({*laTable* ;} *nomFormulaire* ; *imageForm* {; *pageNum*})
  Cette syntaxe vous permet d'obtenir une capture écran d'un "modèle" de formulaire tel qu'affiché dans l'éditeur de formulaires. Tous les objets visibles sont dessinés comme dans l'éditeur ; la commande tient compte des formulaires hérités et des objets placés sur la page 0.
  Si vous souhaitez capturer un formulaire table, passez la table du formulaire dans le paramètre *laTable* puis son nom sous forme de chaîne dans *nomFormulaire*. Pour un formulaire projet, passez directement le nom du formulaire dans *nomFormulaire*.
  Par défaut, la commande capture la page 1 du formulaire. Si vous souhaitez capturer uniquement la page 0 ou une autre page du formulaire, passez son numéro dans le paramètre *pageNum*.

Note : Les deux premiers paramètres de cette commande étant optionnels, il n'est pas possible de passer directement comme argument une fonction retournant un pointeur telle que **Table du formulaire courant**-> ou **Table**->. Cette syntaxe fonctionnera en mode interprété mais sera rejetée lors de la compilation. Il est nécessaire dans ce cas d'utiliser une variable pointeur intermédiaire. Pour plus d'informations, reportez-vous au paragraphe "**Utilisation directe de commandes retournant des pointeurs**".

FORM CHARGER

FORM CHARGER ({laTable ;} formulaire {; *})

Paramètre	Type		Description
laTable	Table	⇒	Table du formulaire à charger (si omis, charger un formulaire projet)
formulaire	Chaîne	⇒	Nom du formulaire (projet ou table) à ouvrir pour l'impression
*	Opérateur	⇒	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

Description

La commande **FORM CHARGER** vous permet de charger le *formulaire* (projet ou table) en mémoire dans le process courant afin d'imprimer des données ou d'analyser son contenu. Il ne peut y avoir qu'un seul formulaire courant par process.

Impression de données

Pour que cette commande puisse être exécutée, une tâche d'impression doit avoir été ouverte au préalable à l'aide de la commande **OUVRIR TACHE IMPRESSION**. La commande **OUVRIR TACHE IMPRESSION** effectue un appel implicite à la commande **FORM LIBERER**, il est donc nécessaire d'exécuter **FORM CHARGER** dans ce contexte. Une fois chargé, le *formulaire* devient le formulaire d'impression courant. Toutes les commandes de gestion des objets, et en particulier la commande **Imprimer objet**, travaillent avec ce formulaire.

Si un formulaire d'impression avait déjà été chargé au préalable (via un appel précédent à la commande **FORM CHARGER**), il est refermé et remplacé par *formulaire*. Vous pouvez ouvrir et refermer plusieurs formulaires dans la même session d'impression. Changer de formulaire d'impression via la commande **FORM CHARGER** ne génère pas de saut de page, il revient au développeur de les gérer.

Seul l'événement formulaire Sur chargement est exécuté durant l'ouverture du formulaire, ainsi que les méthodes des objets du formulaire. Les autres événements formulaire sont ignorés. L'événement formulaire Sur libération est exécuté à l'issue de l'impression.

Pour préserver la cohérence graphique des formulaires, il est conseillé d'appliquer la propriété d'apparence "Impression" sur toutes les plates-formes.

Le formulaire d'impression courant est automatiquement refermé lorsque la commande **FERMER TACHE IMPRESSION** est appelée.

Note de compatibilité : Dans les versions de 4D antérieures à la v14, la commande **FORM CHARGER** (nommée OUVRIR FORMULAIRE IMPRESSION) acceptait une chaîne vide dans le paramètre *formulaire* afin de refermer le formulaire projet courant. Cette syntaxe n'est désormais plus prise en charge et retourne une erreur. Vous devez désormais utiliser la commande **FORM LIBERER** ou la commande **FERMER TACHE IMPRESSION** pour refermer le formulaire.

Analyse du contenu du formulaire

Cette possibilité consiste à charger un formulaire hors-écran à des fins d'analyse. Pour effectuer cette action, il suffit d'appeler **FORM CHARGER** en-dehors d'un contexte de tâche d'impression. Dans ce cas, les événements formulaire ne sont pas exécutés.

FORM CHARGER peut être utilisé avec les commandes **FORM LIRE OBJETS** et **OBJET Lire type** afin d'effectuer tout type de traitement sur le contenu du formulaire. Il est ensuite impératif d'appeler la commande **FORM LIBERER** afin de décharger le formulaire de la mémoire.

A noter que dans tous les cas, le formulaire à l'écran reste chargé (il n'est pas touché par la commande **FORM CHARGER**), il n'est pas nécessaire de le recharger après un **FORM LIBERER**.

Si la commande est exécutée depuis un composant, elle charge par défaut les formulaires du composant. Si vous passez le paramètre *, la méthode chargera les formulaires de la base hôte.

Rappel : Dans le contexte du hors-écran, n'oubliez pas d'appeler **FORM LIBERER** afin d'éviter tout risque de saturation de la mémoire.

Exemple 1

Appel d'un formulaire projet en tâche d'impression :

```
OUVRIR TACHE IMPRESSION
FORM CHARGER("print_form")
    // exécution des événements et des méthodes objet
```

Exemple 2

Appel d'un formulaire table en tâche d'impression :

```
OUVRIR TACHE IMPRESSION
FORM CHARGER([People];"print_form")
    // exécution des événements et des méthodes objet
```

Exemple 3

Analyse du contenu d'un formulaire pour effectuer un traitement sur les zones de saisie de texte :

```
FORM CHARGER([People];"my_form")
    // sélection du formulaire sans exécution des événements ni des méthodes
FORM LIRE OBJETS(tabNomsObj;tabPtrObj;tabPages;*)
Boucle($i;1;Taille tableau(tabNomsObj))
      Si(OBJET Lire type(*;tabNomsObj{$i})=Objet type saisie texte)
          //… traitement
      Fin de si
Fin de boucle
```

```
FORM LIBERER //ne pas oublier
```

FORM DERNIERE PAGE

FORM DERNIERE PAGE
Ne requiert pas de paramètre

Description

La commande **FORM DERNIERE PAGE** change la page courante d'un formulaire pour afficher la dernière page du formulaire. Si aucun formulaire n'est affiché ou chargé via la commande **FORM CHARGER**, ou si la dernière page du formulaire est déjà affichée, **FORM DERNIERE PAGE** ne fait rien.

Exemple

Cet exemple est une méthode appelée par une commande de menu. Elle affiche la dernière page du formulaire courant :

```
FORM DERNIERE PAGE
```

FORM FIXER ENTREE

FORM FIXER ENTREE ({laTable ;} formulaire {; formUtilisateur}{; *})

Paramètre	Type		Description
laTable	Table	⇒	Table pour laquelle définir le formulaire entrée ou Table par défaut si ce paramètre est omis
formulaire	Chaîne	⇒	Nom du formulaire à utiliser
formUtilisateur	Chaîne	⇒	Nom du formulaire utilisateur à utiliser
*		⇒	Taille de fenêtre automatique

Description

FORM FIXER ENTREE désigne *formulaire* ou *formUtilisateur* comme formulaire entrée courant de *laTable* pour le process courant. *formulaire* doit appartenir à *laTable*.

La portée de cette commande est le process courant. Chaque table dispose d'un formulaire entrée courant pour chaque process.

Note : Pour des raisons structurelles, cette commande n'est pas compatible avec les formulaires projets. Si vous passez un formulaire projet dans *formulaire*, la commande ne fait rien.

La commande **FORM FIXER ENTREE** n'affiche pas de formulaire ; elle désigne juste celui qui sera affiché ou utilisé par d'autres commandes. Pour plus d'informations sur la création de formulaires, reportez-vous au manuel *Mode Développement* de 4D.

Pour chaque table de la base, un formulaire entrée par défaut est défini dans la fenêtre de l'Explorateur. Ce formulaire est utilisé si la commande **FORM FIXER ENTREE** n'est pas appelée, ou si le paramètre *formulaire* est invalide.

Le paramètre facultatif *formUtilisateur* permet de désigner un formulaire utilisateur (issu du *formulaire*) comme formulaire entrée par défaut. Si vous passez un nom de formulaire utilisateur valide, ce formulaire sera utilisé par défaut en lieu et place du formulaire entrée dans le process courant. Ce principe permet de disposer simultanément de différents formulaires utilisateurs personnalisés (générés à l'aide de la commande **CREER FORMULAIRE UTILISATEUR**) et d'utiliser celui qui convient en fonction du contexte.
Pour plus d'informations sur les formulaires utilisateurs, reportez-vous à la section **Présentation des formulaires utilisateurs**.

Le formulaire entrée est affiché par de nombreuses commandes. Ces commandes sont généralement utilisées pour la saisie ou la modification de valeurs. Les commandes suivantes affichent un formulaire entrée :

- **AJOUTER ENREGISTREMENT**
- **AFFICHER ENREGISTREMENT**
- **MODIFIER ENREGISTREMENT**
- **CHERCHER PAR EXEMPLE**

Les commandes **VISUALISER SELECTION** et **MODIFIER SELECTION** affichent une liste d'enregistrements dans le formulaire sortie. Chacune d'entre elles permet ensuite à l'utilisateur de double-cliquer sur un enregistrement, qui s'affiche alors dans le formulaire entrée.

Le formulaire entrée est aussi utilisé par les commandes d'import **IMPORTER TEXTE**, **LECTURE SYLK** et **LECTURE DIF**.

Le paramètre optionnel * est destiné à être utilisé conjointement avec les propriétés du formulaire, que vous définissez en mode Développement dans la fenêtre des Propriétés du formulaire, et avec la commande **Creer fenetre**. En passant le paramètre *, vous indiquez à 4D d'utiliser les propriétés du formulaire pour redimensionner automatiquement la fenêtre lors de l'utilisation ultérieure de la fenêtre comme formulaire entrée ou comme dialogue. Reportez-vous à la description de la commande **Creer fenetre** pour plus d'informations sur ce point.

Note : Que vous passiez ou non le paramètre *, **FORM FIXER ENTREE** change le formulaire entrée pour la table.

Exemple 1

L'exemple suivant illustre une utilisation typique de **FORM FIXER ENTREE**. A noter que, si dans cet exemple **FORM FIXER ENTREE** est appelé juste avant que le formulaire soit utilisé, cela n'est absolument pas nécessaire. **FORM FIXER ENTREE** peut en fait être exécuté dans une tout autre méthode, du moment qu'elle est exécutée avant celle-ci :

```
FORM FIXER ENTREE([Sociétés];"Nouvelle Sté") ` Formulaire pour les nouvelles sociétés
AJOUTER ENREGISTREMENT([Sociétés]) ` Ajout d'une nouvelle société
```

Exemple 2

Dans une base de facturation gérant plusieurs sociétés, la création d'une facture doit s'effectuer dans le formulaire utilisateur correspondant :

```
Au cas ou
   :(société="4D SAS")
      FORM FIXER ENTREE([Factures];"Saisie";"4D_SAS")
   :(société="4D Inc")
      FORM FIXER ENTREE([Factures];"Saisie";"4D_Inc")
   :(société="Acme")
      FORM FIXER ENTREE([Factures];"Saisie";"ACME")
Fin de cas
AJOUTER ENREGISTREMENT([Factures])
```

FORM FIXER REDIMENSIONNEMENT HORIZONTAL

FORM FIXER REDIMENSIONNEMENT HORIZONTAL (redimension {; largeurMini {; largeurMaxi}})

Paramètre	Type		Description
redimension	Booléen	⇒	Vrai : le formulaire est redimensionnable horizontalement Faux : le formulaire n'est pas redimensionnable horizontalement
largeurMini	Entier long	⇒	Largeur minimale du formulaire (pixels)
largeurMaxi	Entier long	⇒	Largeur maximale du formulaire (pixels)

Description

La commande **FORM FIXER REDIMENSIONNEMENT HORIZONTAL** permet de modifier par programmation les propriétés de redimensionnement horizontal du formulaire courant. Par défaut, ces propriétés sont définies dans l'éditeur de formulaires en mode Développement. Les nouvelles propriétés sont fixées pour le process courant, elles ne sont pas stockées avec le formulaire.

Le paramètre *redimension* permet de définir si le formulaire est redimensionnable horizontalement, c'est-à-dire si sa largeur est modifiable (manuellement par l'utilisateur ou par programmation).

Si vous passez Vrai, la largeur du formulaire peut être modifiée par l'utilisateur ; 4D utilise comme bornes les valeurs éventuellement passées dans les paramètres *largeurMini* et *largeurMaxi*.

Si vous passez Faux, la largeur courante du formulaire n'est pas modifiable ; dans ce cas, il est inutile de passer des valeurs dans les paramètres *largeurMini* et *largeurMaxi*.

Si vous avez passé Vrai dans le premier paramètre, vous pouvez passer dans les paramètres facultatifs *largeurMini* et *largeurMaxi* les nouvelles largeurs minimale et maximale du formulaire en pixels. Si vous omettez ces paramètres, les valeurs définies en mode Développement (le cas échéant) seront utilisées.

Exemple

Reportez-vous à l'exemple de la commande **FORM FIXER TAILLE**.

FORM FIXER REDIMENSIONNEMENT VERTICAL

FORM FIXER REDIMENSIONNEMENT VERTICAL (redimension {; hauteurMini {; hauteurMaxi}})

Paramètre	Type		Description
redimension	Booléen	⇒	Vrai : le formulaire est redimensionnable verticalement Faux : le formulaire n'est pas redimensionnable verticalement
hauteurMini	Entier long	⇒	Hauteur minimale du formulaire (pixels)
hauteurMaxi	Entier long	⇒	Hauteur maximale du formulaire (pixels)

Description

La commande **FORM FIXER REDIMENSIONNEMENT VERTICAL** permet de modifier par programmation les propriétés de redimensionnement vertical du formulaire courant. Par défaut, ces propriétés sont définies dans l'éditeur de formulaires en mode Développement. Les nouvelles propriétés sont fixées pour le process courant, elles ne sont pas stockées avec le formulaire.

Le paramètre *redimension* permet de définir si le formulaire est redimensionnable verticalement, c'est-à-dire si sa hauteur est modifiable (manuellement par l'utilisateur ou par programmation).

Si vous passez Vrai, la hauteur du formulaire peut être modifiée par l'utilisateur ; 4D utilise comme bornes les valeurs éventuellement passées dans les paramètres *hauteurMini* et *hauteurMaxi*.

Si vous passez Faux, la hauteur courante du formulaire n'est pas modifiable ; dans ce cas, il est inutile de passer des valeurs dans les paramètres *hauteurMini* et *hauteurMaxi*.

Si vous avez passé Vrai dans le premier paramètre, vous pouvez passer dans les paramètres facultatifs *hauteurMini* et *hauteurMaxi* les nouvelles hauteurs minimale et maximale du formulaire en pixels. Si vous omettez ces paramètres, les valeurs définies en mode Développement (le cas échéant) seront utilisées.

Exemple

Reportez-vous à l'exemple de la commande **FORM FIXER TAILLE**.

FORM FIXER SORTIE

FORM FIXER SORTIE ({laTable ;} formulaire {; formUtilisateur})

Paramètre	Type		Description
laTable	Table	⇒	Table pour laquelle définir le formulaire sortie ou Table par défaut si ce paramètre est omis
formulaire	Chaîne	⇒	Nom du formulaire
formUtilisateur	Chaîne	⇒	Nom du formulaire utilisateur à utiliser

Description

FORM FIXER SORTIE vous permet de définir *formulaire* ou *formUtilisateur* comme formulaire sortie courant de *laTable* pour le process courant. *formulaire* doit appartenir à *laTable*.

La portée de cette commande est le process courant. Chaque table dispose de son propre formulaire sortie dans chaque process.

Note : Pour des raisons structurelles, cette commande n'est pas compatible avec les formulaires projets. Si vous passez un formulaire projet dans *formulaire*, la commande ne fait rien.

La commande **FORM FIXER SORTIE** ne provoque pas l'affichage du formulaire ; elle désigne simplement le formulaire devant être imprimé, affiché, ou utilisé par d'autres commandes. Pour plus d'informations sur la création de formulaires, reportez-vous au manuel *Mode Développement* de 4D.

Le formulaire sortie par défaut est défini dans la fenêtre de l'Explorateur pour chaque table. Il est identifié par la lettre S placée près de son nom dans l'Explorateur et dans les boîtes de dialogue listant les formulaires. Ce formulaire par défaut sera utilisé si vous n'appelez pas la commande **FORM FIXER SORTIE** ou si vous passez à cette commande un nom de formulaire erroné ou inexistant.

Le paramètre facultatif *formUtilisateur* permet de désigner un formulaire utilisateur (issu du *formulaire*) comme formulaire sortie par défaut. Si vous passez un nom de formulaire utilisateur valide, ce formulaire sera utilisé par défaut en lieu et place du formulaire sortie dans le process courant. Ce principe permet de disposer simultanément de différents formulaires utilisateurs personnalisés (générés à l'aide de la commande **CREER FORMULAIRE UTILISATEUR**) et d'utiliser celui qui convient en fonction du contexte.

Pour plus d'informations sur les formulaires utilisateurs, reportez-vous à la section **Présentation des formulaires utilisateurs**.

Les formulaires sortie sont exploités par trois groupes de commandes. Le premier groupe gère l'affichage des enregistrements à l'écran, le deuxième gère la génération d'états et le troisième gère l'export de données.

Chacune des commandes suivantes affiche une liste d'enregistrements dans un formulaire sortie :

- **VISUALISER SELECTION**
- **MODIFIER SELECTION**

Vous utilisez le formulaire sortie lorsque vous créez des états à l'aide des commandes suivantes :

- **IMPRIMER ETIQUETTES**
- **IMPRIMER SELECTION**

Chacune des commandes d'export suivantes utilise également le formulaire sortie :

- **ECRITURE DIF**
- **ECRITURE SYLK**
- **EXPORTER TEXTE**

Exemple

L'exemple suivant illustre une utilisation typique **FORM FIXER SORTIE**. Notez que, bien que dans cet exemple la commande **FORM FIXER SORTIE** soit placée juste avant que le formulaire soit utilisé, cela n'est pas obligatoire. En fait, la commande pourrait se trouver dans n'importe quelle autre méthode, dans la mesure où elle est exécutée avant celle-ci :

```
FORM FIXER ENTREE([Parties];"Saisie Parties") ` Sélection du formulaire entrée
FORM FIXER SORTIE([Parties];"Liste Parties") ` Sélection du formulaire sortie
MODIFIER SELECTION([Parties]) ` Cette commande utilise les deux formulaires
```

FORM FIXER TAILLE

FORM FIXER TAILLE ({objet ;} horizontal ; vertical {; *})

Paramètre	Type		Description
objet	Chaîne	⇒	Nom d'objet indiquant les limites du formulaire
horizontal	Entier long	⇒	Si * passé : marge horizontale (pixels) Si * omis : largeur (pixels)
vertical	Entier long	⇒	Si * passé : marge verticale (pixels) Si * omis : hauteur (pixels)
*	Opérateur	⇒	• Si passé, utiliser horizontal et vertical comme marges du formulaire • Si omis, utiliser horizontal et vertical comme largeur et hauteur du formulaire Ce paramètre ne peut pas être passé si objet est passé

Description

La commande **FORM FIXER TAILLE** permet de modifier par programmation la taille du formulaire courant. La nouvelle taille est définie pour le process courant, elle n'est pas stockée avec le formulaire.

Comme en mode Développement, cette commande permet de définir la taille d'un formulaire de trois manières :

- automatiquement — 4D détermine la taille du formulaire sur le principe que tous les objets doivent être visibles — en ajoutant éventuellement une marge horizontale et une marge verticale,
- sur la base de l'emplacement d'un objet du formulaire auquel s'ajoutent éventuellement une marge horizontale et une marge verticale,
- en saisissant des dimensions "absolues" (largeur et hauteur).
  Pour plus d'informations sur les possibilités de dimensionnement des formulaires, reportez-vous au manuel Mode Développement de 4D.

Taille automatique
Pour que le formulaire ait une taille automatique, vous devez utiliser la syntaxe suivante :

```
FORM FIXER TAILLE(horizontal;vertical;*)
```

Dans ce cas, vous devez passer dans *horizontal* et *vertical* les marges (en pixels) que vous souhaitez ajouter à droite et en bas du formulaire.

Taille basée sur un objet
Pour que la taille du formulaire soit basée sur un objet, vous devez utiliser la syntaxe suivante :

```
FORM FIXER TAILLE(objet;horizontal;vertical)
```

Dans ce cas, vous devez passer dans *horizontal* et *vertical* les marges (en pixels) que vous souhaitez ajouter à droite et en bas de l'objet. Il n'est pas possible de passer le paramètre *.

Taille en valeur absolue
Pour passer une taille de formulaire absolue, vous devez utiliser la syntaxe suivante :

```
FORM FIXER TAILLE(horizontal;vertical)
```

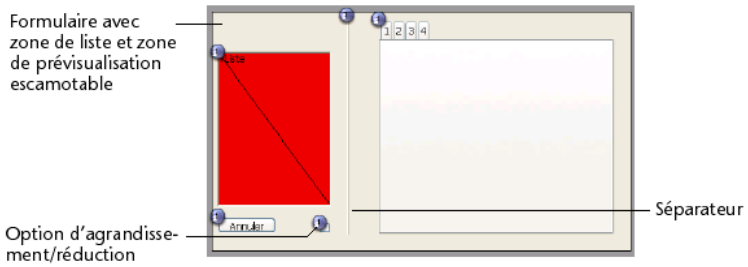
Dans ce cas, vous devez passer dans *horizontal* et *vertical* la largeur et la hauteur (en pixels) du formulaire.

La commande **FORM FIXER TAILLE** modifie la taille du formulaire mais tient compte de ses propriétés de redimensionnement. Par exemple, si la largeur minimale du formulaire est de 500 pixels et si la commande définit une largeur de 400 pixels, la nouvelle largeur du formulaire sera de 500 pixels.

A noter également que cette commande ne modifie pas la taille de la fenêtre du formulaire (il est possible de redimensionner un formulaire sans que la taille de la fenêtre soit modifiée, et inversement). Pour modifier la taille de la fenêtre d'un formulaire, reportez-vous à la description de la commande **REDIMENSIONNER FENETRE FORMULAIRE**.

Exemple

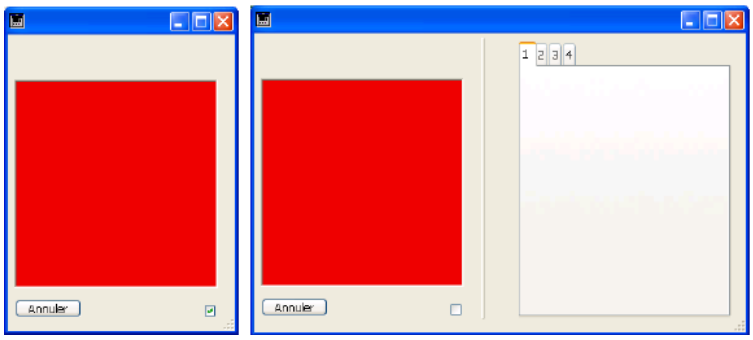
Voici un exemple de mise en place d'une fenêtre de type Explorateur. Le formulaire suivant est défini en mode Développement :



La taille du formulaire est "automatique".
La fenêtre est affichée via l'instruction suivante :

```
$ref:=Creer fenetre formulaire([Table 1];"Form1";Form fenêtre standard;Centrée horizontalement;Centrée
verticalement;*)
DIALOGUE([Table 1];"Form1")
FERMER FENETRE
```

La partie droite de la fenêtre peut être affichée ou masquée via un clic sur l'option d'agrandissement/réduction :



La méthode objet associée à ce bouton est la suivante :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      C_BOOLEEN(b1;<>contracté)
      C_ENTIER LONG(marge)
      marge:=15
      b1:=<>contracté
      Si(<>contracté)
         FORM FIXER REDIMENSIONNEMENT HORIZONTAL(Faux)
         FORM FIXER TAILLE("b1";marge;marge)
      Sinon
         FORM FIXER REDIMENSIONNEMENT HORIZONTAL(Vrai)
         FORM FIXER TAILLE("onglet";marge;marge)
      Fin de si

   :(Evenement formulaire=Sur clic)
      <>contracté:=b1
      Si(b1)
 `contracté
         OBJET LIRE COORDONNEES(*;"b1";$g;$h;$d;$b)
         COORDONNEES FENETRE($gf;$hf;$df;$bf;Fenetre formulaire courant)
         CHANGER COORDONNEES FENETRE($gf;$hf;$gf+$d+marge;$hf+$b+marge;Fenetre formulaire courant)
         FORM FIXER REDIMENSIONNEMENT HORIZONTAL(Faux)
         FORM FIXER TAILLE("b1";marge;marge)

      Sinon
 `déployé
         OBJET LIRE COORDONNEES(*;"onglet";$g;$h;$d;$b)
         COORDONNEES FENETRE($gf;$hf;$df;$bf;Fenetre formulaire courant)
         CHANGER COORDONNEES FENETRE($gf;$hf;$gf+$d+marge;$hf+$b+marge;Fenetre formulaire courant)
         FORM FIXER REDIMENSIONNEMENT HORIZONTAL(Vrai)
         FORM FIXER TAILLE("onglet";marge;marge)
      Fin de si

Fin de cas
```

FORM LIBERER

FORM LIBERER
Ne requiert pas de paramètre

Description

La commande **FORM LIBERER** permet de décharger de la mémoire le formulaire courant désigné via la commande **FORM CHARGER**.

L'appel de cette commande est nécessaire lors de l'utilisation de la commande **FORM CHARGER** hors contexte d'impression (en cas d'impression, le formulaire courant est automatiquement refermé lorsque la commande **FERMER TACHE IMPRESSION** est appelée).

FORM LIRE OBJETS

FORM LIRE OBJETS (tabObjets {; tabVariables {; tabPages}} {; optionPage | *})

Paramètre	Type		Description
tabObjets	Tableau chaîne	⇐	Noms des objets du formulaire
tabVariables	Tableau pointeur	⇐	Pointeurs sur les variables ou champs associés aux objets
tabPages	Tableau entier	⇐	Numéro de page de chaque objet
optionPage | *	Entier long, Opérateur	⇒	1=Page courante du formulaire, 2=Toutes les pages, 4=Pages héritées Si * passé (obsolète) = page courante avec objets hérités

Description

La commande **FORM LIRE OBJETS** retourne sous forme de tableau(x) la liste de tous les objets présents dans le formulaire courant. Cette liste peut être restreinte à la page courante du formulaire et peut exclure les objets des formulaires hérités. La commande peut être utilisée avec les formulaires entrée et sortie.

Si un tableau passé en paramètre n'est pas préalablement déclaré, la commande le crée et le dimensionne automatiquement. Toutefois, dans la perspective de la compilation de l'application, il est recommandé de déclarer explicitement chaque tableau.

Passez dans *tabObjets* le nom du tableau texte devant être rempli avec les noms des objets (chaque nom d'objet est unique au sein d'un formulaire). L'ordre dans lequel les objets apparaissent dans le tableau n'est pas significatif.

Les autres tableaux remplis facultativement par la commande sont synchronisés avec le premier.

Passez dans le paramètre facultatif *tabVariables* le nom du tableau de pointeurs devant être rempli avec des pointeurs vers les variables ou champs associés aux objets. Si un objet n'a pas de variable associée, le pointeur **Nil** est retourné. Dans le cas d'un objet de type "sous-formulaire", un pointeur sur la table du sous-formulaire est retourné.

Le troisième tableau (facultatif), *tabPages*, est rempli avec les numéros de pages du formulaire. Chaque ligne de ce tableau contient le numéro de la page sur laquelle se trouve l'objet correspondant.

Le paramètre optionnel *optionPage* vous permet de désigner la ou les partie(s) du formulaire dont vous souhaitez lire les objets. Par défaut, si le paramètre *optionPage* est omis (ainsi que le paramètre *), les objets de toutes les pages, y compris les objets hérités, sont retournés. Pour délimiter la portée de la commande, vous pouvez passer une (ou une combinaison) des constantes suivantes du thème "**Objets de formulaire (Accès)**" dans le paramètre *optionPage* :

Constante	Type	Valeur	Comment
Form hérité	Entier long	4	Retourne uniquement les objets hérités
Form page courante	Entier long	1	Retourne tous les objets de la page courante, y compris ceux de la page 0, mais exclut les objets hérités
Form toutes les pages	Entier long	2	Retourne tous les objets de toutes les pages, mais exclut les objets hérités

Note de compatibilité : Passer le paramètre * équivaut à passer Form page courante+Form hérité. Cependant, la syntaxe utilisant le paramètre * est obsolète et ne doit plus être utilisée.

Exemple 1

Vous souhaitez obtenir les objets de toutes les pages, y compris ceux des formulaires hérités (le cas échéant):

```
 //Formulaire ouvert
FORM LIRE OBJETS(tabObjets;tabVariables;tabPages)
```

Ou :

```
 //Formulaire chargé
FORM CHARGER([Table1];"MonForm")
FORM LIRE OBJETS(tabObjets;tabVariables;tabPages;Form toutes les pages+Form hérité)
```

Exemple 2

Vous souhaitez obtenir les objets de la page courante du formulaire chargé, incluant la page 0 de ce formulaire ainsi que les objets des formulaires hérités (le cas échéant) :

```
FORM CHARGER("MonForm")
FORM ALLER A PAGE(2)
FORM LIRE OBJETS(tabObjets;tabVariables;tabPages;Form page courante+Form hérité)
```

Exemple 3

Vous souhaitez obtenir les objets des formulaires hérités. S'il n'y a pas de formulaire hérité, les tableaux seront retournés vides.

```
FORM CHARGER("MonForm")
FORM LIRE OBJETS(tabObjets;tabVariables;tabPages;Form hérité)
```

Exemple 4

Vous souhaitez obtenir les objets de la page 4, ainsi que ceux de la page 0, mais pas ceux des formulaires hérités (le cas échéant) :

```
FORM CHARGER([Table1];"MyForm")
FORM ALLER A PAGE(4)
FORM LIRE OBJETS(tabObjets;tabVariables;tabPages;Form page courante)
```

Exemple 5

Vous souhaitez obtenir les objets de toutes les pages, mais sans ceux des formulaires hérités :

```
FORM CHARGER([Table1];"MonForm")
FORM LIRE OBJETS(tabObjets;tabVariables;tabPages;Form toutes les pages)
```

FORM Lire page courante

FORM Lire page courante {(*)} -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Retourner le numéro de la page du sous-formulaire courant
Résultat	Entier long	⮌	Numéro de la page courante du formulaire courant

Description

FORM Lire page courante retourne le numéro de la page courante du formulaire actuellement affiché ou du formulaire courant chargé via la commande **FORM CHARGER**.

Le paramètre * est utile lorsque la commande est appelée dans le contexte d'un sous-formulaire en page contenant plusieurs pages. Dans ce cas, si vous passez ce paramètre, la commande retourne le numéro de la page courante du sous-formulaire courant (celui qui a appelé la commande). Par défaut, si le paramètre * est omis, la commande s'applique toujours au formulaire parent.

Exemple

Alors que vous êtes en train d'utiliser un formulaire, si vous choisissez une commande de menu ou si le formulaire reçoit un appel d'un autre process, vous voulez que des actions différentes soient effectuées en fonction de la page du formulaire affichée. Vous pouvez alors écrire :

```
   ` Méthode formulaire [maTable];"monFormulaire"
Au cas ou
   :(Evenement formulaire=Sur chargement)
   ` ...
   :(Evenement formulaire=Sur libération)
   ` ...
   :(Evenement formulaire=Sur menu sélectionné)
      $vlNuméroMenu:=Menu choisi>>16
      $vlNuméroCmde:=Menu choisi & 0xFFFF
      Au cas ou
         :($vlNuméroMenu=...)
            Au cas ou
               :($vlNuméroCmde=...)
               :(FORM Lire page courante=1)
   ` Effectuer une action appropriée pour la page 1
               :(FORM Lire page courante=2)
   ` Effectuer une action appropriée pour la page 2
   ` ...
               :($vlNuméroCmde=...)
   ` ...
            Fin de cas
         :($vlNuméroMenu=...)
   ` ...
      Fin de cas
   :(Evenement formulaire=Sur appel extérieur)
      Au cas ou
         :(FORM Lire page courante=1)
   ` Fournir une réponse appropriée pour la page 1
         :(FORM Lire page courante=2)
   ` Fournir une réponse appropriée pour la page 2
      Fin de cas
   ` ...
Fin de cas
```

FORM LIRE PARAMETRE

FORM LIRE PARAMETRE ({laTable ;} formulaire ; sélecteur ; valeur)

Paramètre	Type		Description
laTable	Table	⇒	Table du formulaire ou Table par défaut si ce paramètre est omis
formulaire	Chaîne	⇒	Nom du formulaire
sélecteur	Entier long	⇒	Code du paramètre
valeur	Entier long	⇐	Valeur courante du paramètre

Description

La commande **FORM LIRE PARAMETRE** permet de lire la *valeur* courante d'un paramètre du formulaire désigné par *laTable* et *formulaire*.

sélecteur désigne le paramètre du formulaire dont vous souhaitez connaître la valeur. Vous pouvez utiliser la constante suivante, placée dans le thème **Paramètres de formulaire** :

Constante	Type	Valeur
Objets non inversés	Entier long	0

Lorsque vous utilisez la constante Objets non inversés comme *sélecteur*, la commande retourne dans *valeur* le mode d'affichage réel du formulaire en mode Application sous Windows. Ce paramètre est utilisé dans le cadre du déploiement d'applications dans des langues "de droite à gauche". Pour plus d'informations sur la prise en charge des langues de droite à gauche, reportez-vous manuel *Mode Développement* de 4D.

- si *valeur* contient 0, les objets du formulaire sont inversés,
- si *valeur* contient 1, les objets du formulaire ne sont pas inversés.

Si la commande est appelée en-dehors du contexte du mode Application sous Windows, elle retourne toujours 1.

A noter que l'inversion effective des objets d'un formulaire dépend de la combinaison de plusieurs paramètres : la valeur de la préférence "Inversion des objets en mode Application", la valeur de l'option de formulaire "Ne pas inverser les objets" et le système sur lequel la base est exécutée. Le tableau suivant précise la valeur retournée par la commande **FORM LIRE PARAMETRE** en fonction de ces différentes combinaisons :

Préférences : "Inversion des objets en mode Application" (1)	Propriétés du formulaire :"Ne pas inverser les objets"	Système Windows Droite à gauche	Valeur retournée dans FORM LIRE PARAMETRE
Non	X	X	1
		X	1
	X		1
			1
Automatique	X	X	1
		X	0
	X		1
			1
Oui	X	X	1
		X	0
	X		1
			0

(1) Cette préférence peut également être fixée ou lue via les commandes **FIXER PARAMETRE BASE** et **Lire parametre base**.

FORM LIRE PROPRIETES

FORM LIRE PROPRIETES ({laTable ;} nomForm ; largeur ; hauteur {; nbPages {; largeurFixe {; hauteurFixe {; titre}}}})

Paramètre	Type		Description
laTable	Table	➡	Table du formulaire ou Table par défaut si ce paramètre est omis
nomForm	Chaîne	➡	Nom du formulaire
largeur	Entier long	⬅	Largeur du formulaire (en pixels)
hauteur	Entier long	⬅	Hauteur du formulaire (en pixels)
nbPages	Entier long	⬅	Nombre de pages du formulaire
largeurFixe	Booléen	⬅	Vrai = Largeur fixe, Faux = Largeur variable
hauteurFixe	Booléen	⬅	Vrai = Hauteur fixe, Faux = Hauteur variable
titre	Texte	⬅	Nom de la fenêtre du formulaire

Description

La commande **FORM LIRE PROPRIETES** retourne des propriétés du formulaire *nomForm*.

Les paramètres *largeur* et *hauteur* retournent (en pixels) la largeur et la hauteur du formulaire. Ces valeurs sont déterminées à partir des propriétés de dimensionnement du formulaire :

- Si la taille du formulaire est **automatique**, sa largeur et sa hauteur sont calculées de manière à ce qu'il affiche tous les objets qu'il contient, en tenant compte, le cas échéant, des marges horizontale et verticale qui ont été définies.
- Si la taille du formulaire est **fixe**, sa largeur et sa hauteur sont celles qui ont été saisies manuellement dans les zones correspondantes.
- Si la taille du formulaire est **basée sur un objet**, sa largeur et sa hauteur sont calculées par rapport à la position de cet objet.

Le paramètre *nbPages* retourne le nombre de pages du formulaire, page 0 (zéro) non comprise.

Les paramètres *largeurFixe* et *hauteurFixe* indiquent si la largeur et la hauteur du formulaire sont fixes (le paramètre contient **Vrai**) ou redimensionnables (le paramètre contient **Faux**).

Le paramètre *titre* retourne le nom de la fenêtre du formulaire, tel qu'il a été défini dans la Liste des propriétés en mode Développement. Si aucun nom n'a été défini, le paramètre *titre* contient une chaîne vide.

FORM LIRE REDIMENSIONNEMENT HORIZONTAL

FORM LIRE REDIMENSIONNEMENT HORIZONTAL (redimension {; largeurMini {; largeurMaxi}})

Paramètre	Type		Description
redimension	Booléen	⇦	Vrai : le formulaire est redimensionnable horizontalement, Faux : le formulaire n'est pas redimensionnable horizontalement
largeurMini	Entier long	⇦	Largeur minimale du formulaire (pixels)
largeurMaxi	Entier long	⇦	Largeur maximale du formulaire (pixels)

Description

La commande **FORM LIRE REDIMENSIONNEMENT HORIZONTAL** retourne dans les variables *redimension*, *largeurMini* et *largeurMaxi* les propriétés de redimensionnement horizontal du formulaire courant. Ces propriétés peuvent avoir été définies pour le formulaire dans l'éditeur de formulaires en mode Développement ou pour le process courant via la commande **FORM FIXER REDIMENSIONNEMENT HORIZONTAL**.

FORM LIRE REDIMENSIONNEMENT VERTICAL

FORM LIRE REDIMENSIONNEMENT VERTICAL (redimension {; hauteurMini {; hauteurMaxi}})

Paramètre	Type		Description
redimension	Booléen	⇦	Vrai : le formulaire est redimensionnable verticalement, Faux : le formulaire n'est pas redimensionnable verticalement
hauteurMini	Entier long	⇦	Hauteur minimale du formulaire (pixels)
hauteurMaxi	Entier long	⇦	Hauteur maximale du formulaire (pixels)

Description

La commande **FORM LIRE REDIMENSIONNEMENT VERTICAL** retourne dans les variables *redimension*, *hauteurMini* et *hauteurMaxi* les propriétés de redimensionnement vertical du formulaire courant. Ces propriétés peuvent avoir été définies pour le formulaire dans l'éditeur de formulaires en mode Développement ou pour le process courant via la commande **FORM FIXER REDIMENSIONNEMENT HORIZONTAL**.

FORM PAGE PRECEDENTE

FORM PAGE PRECEDENTE
Ne requiert pas de paramètre

Description

FORM PAGE PRECEDENTE change la page courante d'un formulaire pour afficher la page précédente. Si aucun formulaire n'est affiché ou chargé via la commande **FORM CHARGER**, ou si la page affichée est la première page du formulaire, **FORM PAGE PRECEDENTE** ne fait rien.

Exemple

Cet exemple est une méthode appelée par une commande de menu. Elle provoque l'affichage la page qui précède celle qui est actuellement affichée :

```
FORM PAGE PRECEDENTE
```

FORM PAGE SUIVANTE

FORM PAGE SUIVANTE
Ne requiert pas de paramètre

Description

FORM PAGE SUIVANTE change la page courante d'un formulaire pour afficher la page suivante. Si aucun formulaire n'est affiché ou chargé via la commande **FORM CHARGER**, ou si la page affichée est la dernière page du formulaire, **FORM PAGE SUIVANTE** ne fait rien.

Exemple

Cet exemple est une méthode appelée par une commande de menu. Elle provoque l'affichage la page qui suit celle qui est actuellement affichée :

```
FORM PAGE SUIVANTE
```

FORM PREMIERE PAGE

FORM PREMIERE PAGE
Ne requiert pas de paramètre

Description

La commande **FORM PREMIERE PAGE** change la page courante d'un formulaire pour afficher la première page du formulaire. Si aucun formulaire n'est affiché ou chargé via la commande **FORM CHARGER**, ou si la première page du formulaire est déjà affichée, **FORM PREMIERE PAGE** ne fait rien.

Exemple

Cet exemple est une méthode appelée par une commande de menu. Elle affiche la première page du formulaire :

```
FORM PREMIERE PAGE
```

Nom du formulaire courant

Nom du formulaire courant -> Résultat

Paramètre	Type		Description
Résultat	Texte		Nom du formulaire projet courant ou du formulaire table courant dans le process

Description

La commande **Nom du formulaire courant** retourne le nom du formulaire courant défini pour le process. Le formulaire courant peut être un formulaire projet ou un formulaire table.

Par défaut, si vous n'avez pas appelé la commande **FORM CHARGER** dans le process courant, le formulaire courant est le formulaire en cours d'affichage ou d'impression. Si vous avez appelé la commande **FORM CHARGER** dans le process, le formulaire courant reste celui défini par cette commande jusqu'à l'appel de **FORM LIBERER** (ou **FERMER TACHE IMPRESSION**).

S'il n'y a pas de formulaire courant défini pour le process, la commande retourne une chaîne vide.

Exemple 1

Dans un formulaire de saisie, vous placez le code suivant dans un bouton :

```
C_TEXTE($FormName)
$fen:=Creer fenetre formulaire([Adhérents];"Entrée";Form fenêtre standard)
DIALOGUE([Adhérents];"Entrée")
$FormName:=Nom du formulaire courant
    // $FormName = "Entrée"
FORM CHARGER([Adhérents];"Drag")
$FormName:=Nom du formulaire courant
    // $FormName = "Drag"
    //...
```

Exemple 2

Vous souhaitez obtenir le formulaire courant si c'est un formulaire projet :

```
$PointerTable:=Table du formulaire courant
Si(Nil($PointerTable)) //il s'agit d'un formulaire projet
      $FormName:=Nom du formulaire courant
      ... // traitement
Fin de si
```

Formulaires utilisateurs

- Présentation des formulaires utilisateurs
- CREER FORMULAIRE UTILISATEUR
- LISTE FORMULAIRES UTILISATEURS
- MODIFIER FORMULAIRE
- SUPPRIMER FORMULAIRE UTILISATEUR

Présentation des formulaires utilisateurs

Dans 4D, le développeur peut proposer aux utilisateurs de créer ou de modifier des formulaires personnalisés. Ces “formulaires utilisateurs” sont alors utilisables pour l'affichage, la saisie, etc., comme n’importe quel formulaire de 4D.

Principes d'utilisation

Un formulaire utilisateur est basé sur un formulaire 4D standard créé par le développeur en mode Développement (appelé formulaire “source” ou formulaire “développeur”), auquel la propriété **Modifiable par l’utilisateur** a été appliquée dans l'éditeur de formulaires. Un éditeur de formulaires simplifié (appelé par la commande **MODIFIER FORMULAIRE**) permet aux utilisateurs de modifier l’apparence du formulaire, d’ajouter des objets graphiques (via une bibliothèque d'objets spécifique), de masquer des éléments, etc. — le développeur peut contrôler les actions autorisées.

Les formulaires utilisateurs peuvent être employés de deux manières différentes :

- L’utilisateur modifie le formulaire “source” pour l’adapter à ses besoins à l’aide de la commande **MODIFIER FORMULAIRE**. Le formulaire utilisateur est conservé en local et est utilisé automatiquement à la place du formulaire original.

  Ce fonctionnement répond aux besoins pour le développeur de paramétrer sur site des boîtes de dialogue, par exemple pour coller le logo de l’entreprise dans les formulaires, masquer des champs inutiles, etc.

- Le formulaire “source” sert de modèle de base que l’utilisateur peut dupliquer à loisir pour générer autant de copies que nécessaire (via la commande **CREER FORMULAIRE UTILISATEUR**). Chaque copie est librement paramétrable (contenu, nom, etc.) à l’aide de la commande **MODIFIER FORMULAIRE**. Le nom de chaque formulaire utilisateur doit simplement être unique. Les commandes **FORM FIXER ENTREE** et **FORM FIXER SORTIE** permettent ensuite de désigner le formulaire utilisateur à utiliser dans chaque process.

  Ce fonctionnement répond par exemple aux besoins de génération d’états personnalisés.

Stockage et mise à jour des formulaires utilisateurs

Les mécanismes des formulaires utilisateurs fonctionnent avec les bases compilées et interprétées, avec 4D en mode local, 4D Server ou 4D Desktop. En mode client/serveur, les formulaires modifiés par l’utilisateur sont disponibles sur tous les postes.

4D assure automatiquement la gestion des modifications des formulaires. Lorsqu’un formulaire est déclaré **Modifiable par l’utilisateur**, il est verrouillé en mode Développement. Le développeur doit explicitement cliquer sur l’icône de déverrouillage afin de pouvoir accéder aux objets du formulaire. Cette opération rend automatiquement obsolètes les formulaires utilisateurs liés, qui devront alors être regénérés. Lorsqu’un formulaire “source” est supprimé, les formulaires utilisateurs liés sont supprimés.

Les formulaires utilisateurs sont stockés dans un fichier indépendant suffixé .4DA, placé à côté du fichier de structure principal (.4DB / .4DC). Ce fichier est appelé “fichier de structure utilisateur”. Le fonctionnement de ce fichier est transparent : 4D utilise un formulaire utilisateur lorsqu’il existe (la commande **LISTE FORMULAIRES UTILISATEURS** permet de connaître à tout moment les formulaires utilisateurs valides). C'est également dans ce fichier que les commandes **FORM FIXER ENTREE** et **FORM FIXER SORTIE** recherchent les formulaires utilisateurs.

Lorsqu’un formulaire utilisateur est obsolète, il est supprimé et 4D utilise par défaut le formulaire source.

En client/serveur, le fichier .4DA est distribué sur les postes clients suivant les mêmes règles que le fichier de structure principal.

Ce principe permet de conserver les formulaires utilisateurs non obsolètes en cas de mise à jour de la structure par le développeur.

Codes d'erreurs

Des codes d'erreurs spécifiques peuvent être retournés lors de l'utilisation des commandes de gestion des formulaires utilisateurs. Ces codes, situés dans l'intervalle -9750 à -9759, sont décrits dans la section **Erreurs de la base de données (-10602 -> 4004)**.

Formulaires utilisateurs et formulaires projet

Les mécanismes des formulaires utilisateurs ne sont pas compatibles avec les formulaires projet. Les commandes du thème “Formulaires utilisateurs” ne peuvent donc pas être utilisées avec les formulaires projet.

CREER FORMULAIRE UTILISATEUR

CREER FORMULAIRE UTILISATEUR (laTable ; formulaire ; formUtilisateur)

Paramètre	Type		Description
laTable	Table	⇒	Table du formulaire source
formulaire	Chaîne	⇒	Nom du formulaire table source
formUtilisateur	Chaîne	⇒	Nom du nouveau formulaire utilisateur

Description

La commande **CREER FORMULAIRE UTILISATEUR** duplique le *formulaire* table 4D dont la table et le nom sont passés en paramètres et crée un nouveau formulaire utilisateur nommé *formUtilisateur*.

Une fois créé, le formulaire *formUtilisateur* pourra être modifié à l'aide de la commande **MODIFIER FORMULAIRE**. Cette commande permet de créer N formulaires utilisateurs (par exemple divers formulaires d'états) à partir d'un même formulaire source.

Variables et ensembles système

La variable OK retourne 1 si l'opération s'est déroulée correctement et 0 sinon.

Gestion des erreurs

Une erreur est générée si :

- *formulaire* est déjà un formulaire utilisateur,
- le nom du formulaire utilisateur *formUtilisateur* est identique à celui du formulaire source ou d'un formulaire utilisateur existant,
- l'utilisateur ne possède pas les droits d'accès adéquats.
  Vous pouvez intercepter ces erreurs à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

LISTE FORMULAIRES UTILISATEURS

LISTE FORMULAIRES UTILISATEURS (laTable ; formulaire ; tabFormUtilisateurs)

Paramètre	Type		Description
laTable	Table	⇨	Table du formulaire source
formulaire	Chaîne	⇨	Nom du formulaire table source
tabFormUtilisateurs	Tableau chaîne	⇦	Noms des formulaires utilisateurs issus du formulaire source

Description

La commande **LISTE FORMULAIRES UTILISATEURS** remplit le tableau *tabFormUtilisateurs* avec les noms des formulaires utilisateurs issus du formulaire développeur (formulaire table) désigné par les paramètres *laTable* et *formulaire*.

Si le formulaire utilisateur a été créé directement à l'aide de la commande **MODIFIER FORMULAIRE**, *tabFormUtilisateurs* contient comme seul élément une chaîne vide ("").

Le tableau est retourné vide si aucun formulaire utilisateur n'existe pour le formulaire développeur spécifié.

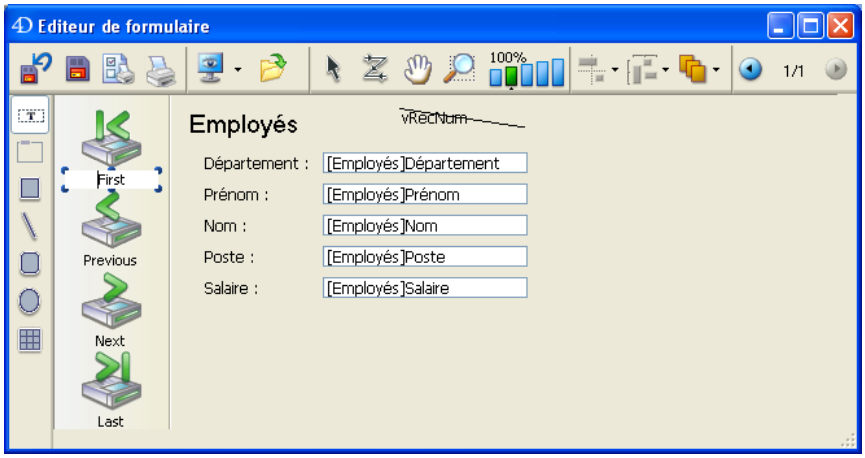
MODIFIER FORMULAIRE

MODIFIER FORMULAIRE (laTable ; formulaire {; formUtilisateur {; bibliothèque}})

Paramètre	Type		Description
laTable	Table	⇒	Table du formulaire à modifier
formulaire	Chaîne	⇒	Nom du formulaire table à modifier
formUtilisateur	Chaîne	⇒	Nom du formulaire utilisateur à modifier
bibliothèque	Chaîne	⇒	Chemin d'accès complet de la bibliothèque d'objets utilisable

Description

La commande **MODIFIER FORMULAIRE** ouvre le formulaire table désigné par les paramètres *laTable*, *formulaire* ainsi que, facultativement, *formUtilisateur* dans l'éditeur de formulaires utilisateurs :



Note : La fenêtre de l'éditeur ne s'ouvre que si elle est la première fenêtre du process. Autrement dit, il sera généralement nécessaire d'ouvrir un nouveau process pour afficher l'éditeur.

Si vous passez une chaîne vide dans le paramètre *formUtilisateur* et s'il n'existe pas déjà un formulaire utilisateur lié à *formulaire*, le formulaire source est affiché dans l'éditeur. Le formulaire modifié est ensuite dupliqué dans le fichier de structure utilisateur (.4DA) et sera utilisé en remplacement de *formulaire*.

Si un formulaire utilisateur avait déjà été généré à partir de *formulaire* à l'aide de cette commande, le formulaire utilisateur s'affiche dans l'éditeur. Si vous souhaitez dans ce cas repartir du formulaire source, vous devez au préalable supprimer le formulaire utilisateur à l'aide de la commande **SUPPRIMER FORMULAIRE UTILISATEUR**.

Le paramètre *formUtilisateur* permet de désigner un formulaire utilisateur (créé à l'aide de la commande **CREER FORMULAIRE UTILISATEUR**) à modifier. Dans ce cas, ce formulaire est affiché dans l'éditeur.

Passez dans le paramètre facultatif *bibliothèque* le chemin d'accès complet de la bibliothèque d'objets que l'utilisateur sera autorisé à utiliser pour personnaliser le formulaire. Lorsqu'elles sont utilisées dans le contexte de l'éditeur de formulaires utilisateurs, les bibliothèques d'objets permettent de coller des objets avec leurs propriétés graphiques et leurs actions automatiques. Les objets auxquels une méthode est associée n'apparaissent pas dans la bibliothèque. Attention, il est du ressort du développeur de vérifier que l'ajout des objets d'une bibliothèque n'est pas incompatible avec le formulaire utilisateur (et ses objets) au niveau des noms, des variables et des types.

En mode client/serveur, la bibliothèque doit se trouver dans le dossier **Resources** de la base de données, au même niveau que le dossier **Plugins**, afin qu'elle soit disponible sur tous les postes clients. Si la bibliothèque est valide, elle est ouverte avec la fenêtre du formulaire.

Pour plus d'informations sur les bibliothèques d'objets, reportez-vous au manuel Mode Développement de la documentation de 4D.

Exemple

Dans cet exemple, l'utilisateur peut choisir une bibliothèque puis modifier un formulaire de dialogue :

```
ASSOCIER TYPES FICHIER("4DLB";"4IL";"Bibliothèque 4D")
$vAbib:=Selectionner document(1;"4DLB";"Veuillez sélectionner une bibliothèque";0)
Si(OK=1)
  `Une bibliothèque a été choisie
   $vACheminLib:=Document
Sinon
   $vACheminLib:=""
Fin de si

MODIFIER FORMULAIRE([Dialogs];"Welcome";"";$vACheminLib)
Si(OK=1)
  `Présentation du formulaire modifié
   DIALOGUE([Dialogs];"Welcome")
Fin de si
```

Variables et ensembles système

Si l'utilisateur sauvegarde les modifications éventuellement effectuées dans l'éditeur, la variable OK prend la valeur 1. En cas d'erreur, OK prend la valeur 0.

Gestion des erreurs

Une erreur est générée si :

- le formulaire n'a pas été déclaré modifiable par l'utilisateur en mode Développement ou n'existe pas,
- le formulaire est déjà ouvert en modification dans un autre process,
- l'utilisateur ne possède pas les droits d'accès adéquats.
  Vous pouvez intercepter ces erreurs à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

SUPPRIMER FORMULAIRE UTILISATEUR

SUPPRIMER FORMULAIRE UTILISATEUR (laTable ; formulaire ; formUtilisateur)

Paramètre	Type		Description
laTable	Table	⇒	Table du formulaire utilisateur
formulaire	Chaîne	⇒	Nom du formulaire table source
formUtilisateur	Chaîne	⇒	Nom du formulaire utilisateur

Description

La commande **SUPPRIMER FORMULAIRE UTILISATEUR** permet de supprimer le formulaire utilisateur désigné par les paramètres *laTable*, *formulaire* et *formUtilisateur*.

Variables et ensembles système

Si le formulaire utilisateur est correctement supprimé, la variable OK retourne 1. Dans le cas contraire, OK prend la valeur 0.

Gestion des erreurs

Une erreur est générée si :

- le formulaire utilisateur n'existe pas ou *formUtilisateur* contient une chaîne vide (-9757),
- l'utilisateur ne possède pas les droits d'accès adéquats.
  Vous pouvez intercepter ces erreurs à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

Formules

- Utiliser des tokens dans les formules
- EDITER FORMULE
- EXECUTER FORMULE
- FIXER METHODES AUTORISEES
- LIRE METHODES AUTORISEES

Utiliser des tokens dans les formules

Présentation

Le langage de 4D comporte un système unique en son genre de *tokenisation* de tous les noms d'objets du langage utilisés dans le code (commandes, tables, champs, constantes, mots-clés). *Tokeniser* ces noms signifie les stocker en interne sous forme de références absolues (des numéros) au moment de leur saisie dans l'éditeur de code et les restituer à l'exécution ou à l'affichage en tenant compte du contexte. Ce principe permet de garantir que le code sera toujours correctement interprété, même si vous renommez vos tables ou vos champs, ou si des commandes du langage 4D sont renommées au fil des versions de l'application.

Note : Ce principe permet également d'assurer la traduction automatique du code lorsque vous avez activé l'option "Utiliser langage français et paramètres régionaux système" de la **Page Méthodes** des Préférences et ouvrez vos bases avec des versions de 4D de langues différentes.

La *tokenisation* est entièrement transparente pour les développeurs 4D lorsqu'ils travaillent dans l'éditeur de code. Toutefois, ce mécanisme n'est pas automatiquement mis en oeuvre dans les formules de 4D, puisqu'elles sont constituées de texte qui est interprété au moment de l'exécution, et non au moment de la saisie. En fait, c'est le cas dès que le code 4D est exprimé sous forme de texte brut, notamment lorsque le code est exporté puis importé via les commandes **METHODE LIRE CODE** et **METHODE FIXER CODE**, copié-collé, ou interprété depuis des **Balises de transformation 4D**.
Pour continuer à bénéficier des mécanismes de la *tokenisation* dans ce contexte, il suffit d'utiliser une syntaxe explicite, consistant à faire précéder les noms des objets du langage de leur *token*. Cette syntaxe est décrite ci-dessous.

Syntaxe tokenisée

Par défaut, le mécanisme des *tokens* n'est pas automatiquement mis en oeuvre dans les formules de 4D (ainsi que dans les contextes où le code 4D est exprimé sous forme de texte brut, cf. ci-dessus). Par conséquent, 4D propose, pour les éléments nommés contenus dans les expressions, une syntaxe spéciale permettant de référencer directement les *tokens* : il suffit d'ajouter à la suite du nom de l'élément un suffixe spécifique indiquant sa nature (commande, champ...) puis sa référence. La **syntaxe tokenisée** est décrite dans ce tableau :

Elément	Exemple (syntaxe standard)	Suffixe	Exemple (syntaxe tokenisée)	Commentaire
Commande 4D	Chaine(a)	:Cxx	String:C10(a)	xx est le numéro de la commande
Table	[Employés]	:xx	[Employés:1]	xx est le numéro de la table
Champ	[Employés]Nom	:xx	[Employés:1]Nom:2	xx est le numéro du champ
Plugin 4D	PV IMPRIMER(zone)	:Pxx:yy	PV IMPRIMER:P13000:229(zone)	xx est l'ID du plug-in et yy l'index de la commande

Note : Les lettres utilisées dans les suffixes (C, P) doivent impérativement être en majuscules, sinon elles ne seront pas correctement interprétées.

Lorsque vous utilisez cette syntaxe, vous avez la garantie que vos formules seront correctement interprétées même en cas de renommage ou d'exécution de la base dans une langue différente.

Note : Les constantes sont également tokenisées dans le langage, toutefois dans les formules il suffit de passer leur valeur afin de les rendre indépendantes du contexte.

Cette syntaxe est acceptée dans toutes les formules 4D (ou expressions 4D), quel que soit leur contexte d'appel :

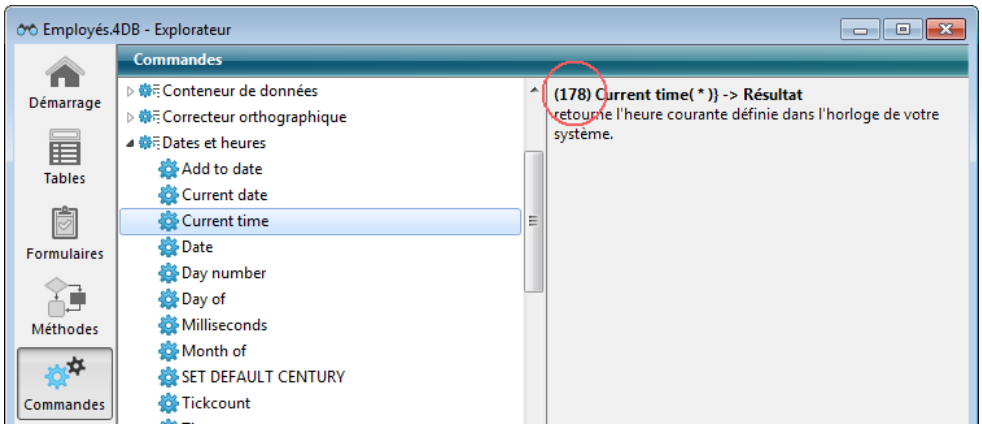
- formules 4D exécutées via l'**Editeur de formules** ou via les commandes telles que **EXECUTER FORMULE**, **APPLIQUER A SELECTION**, **CHERCHER PAR FORMULE**, **LISTBOX INSERER COLONNE FORMULE**, etc.
- expressions insérées dans des zones de texte riche (cf. **ST INSERER EXPRESSION** et **Balises prises en charge**),
- expressions calculées dans les balises de transformation (cf. **Balises de transformation 4D**),
- expressions insérées dans les zones de plug-ins,
- expressions insérées dans des zones 4D Write Pro (à compter de 4D v15).

Où trouver les numéros des éléments ?

La syntaxe tokenisée nécessite l'ajout de numéros de référence des éléments. L'emplacement de ces références dépend de la nature de l'élément.

Commandes 4D

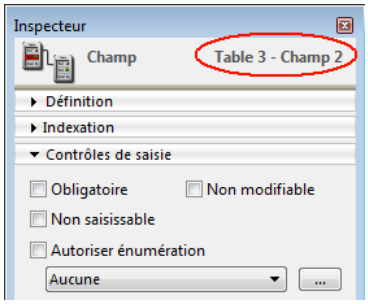
Les numéros des commandes sont indiqués dans ce manuel *Langage* (zone "Propriétés") ainsi que dans la page **Commandes** de l'Explorateur :



Tables et champs

Les numéros des tables et des champs peuvent être obtenus à l'aide des commandes **Table** et **Champ**.

Ils sont également affichés dans la **Palette Inspecteur** de l'éditeur de structure :



Commandes de plug-ins 4D

Pour connaître les tokens des commandes de plug-ins 4D, l'astuce consiste à saisir le code souhaité dans l'éditeur de méthodes, puis à relancer 4D après avoir inactivé le plug-in (par exemple en déplaçant son dossier). Dans l'éditeur de méthodes, seuls les tokens sont affichés ; vous pouvez alors les copier.

Code avec plug-in installé :

```
12    PV FIXER HAUTEUR LIGNES(Zone;1;10;PV Lire hauteur ligne(Zone;10)+$Hauteur)
```

Le même code après inactivation du plug-in :

```
12    Ô13000;75Ô (Zone;1;10;Ô13000;76Ô (Zone;10)+$Hauteur)
```

EDITER FORMULE

EDITER FORMULE (laTable ; formule)

Paramètre	Type		Description
laTable	Table	⇨	Table à afficher par défaut dans l'éditeur de formules
formule	Tableau chaîne	⇨	Variable contenant la formule à afficher dans l'éditeur de formules ou "" pour uniquement afficher l'éditeur
		⇦	Formule validée par l'utilisateur

Description

La commande **EDITER FORMULE** affiche l'éditeur de formules afin de permettre à l'utilisateur d'écrire ou de modifier une formule. L'éditeur contient à l'ouverture :

- dans la liste de gauche, les champs de la table désignée par le paramètre *laTable*,
- dans la zone de formule, la formule contenue dans la variable *formule*. Si vous avez passé une chaîne vide dans *formule*, l'éditeur est affiché sans formule.

L'utilisateur peut modifier la *formule* affichée et la sauvegarder. Il peut également en écrire ou en charger une nouvelle. Dans tous les cas, lorsque l'utilisateur valide la boîte de dialogue, la variable système OK prend la valeur 1 et la variable *formule* contient la formule définie. Si l'utilisateur annule la boîte de dialogue, la variable système OK prend la valeur 0 et *formule* est inchangée.

Notes :

- Par défaut, l'accès aux méthodes et aux commandes est restreint dans l'éditeur de formules pour tous les utilisateurs (sauf, dans les bases de données créées avec 4D 2004.4 et suivantes, pour le Super_Utilisateur et l'Administrateur). Lorsque ce mécanisme est actif, vous devez explicitement désigner les éléments accessibles aux utilisateurs à l'aide de la commande **FIXER METHODES AUTORISEES**. Si la formule fait appel à des méthodes qui n'ont pas été préalablement autorisées, une erreur de syntaxe est générée et il n'est pas possible de valider la boîte de dialogue.
- L'éditeur de formules n'est associé à aucune barre de menus par défaut. L'équivalent d'un menu **Edition** standard doit être installé dans le process appelant si vous souhaitez que l'utilisateur bénéficie des raccourcis couper / copier / coller dans l'éditeur de formules.

A noter qu'au moment de la validation de la boîte de dialogue, la commande n'exécute pas la *formule*, seul le contenu de la variable est validé et mis à jour. Si vous voulez exécuter la *formule*, vous devez utiliser la commande **EXECUTER FORMULE**.

Exemple

Affichage de l'éditeur avec la table [Salaires] et sans formule pré-saisie puis exécution de la formule sur la sélection courante :

```
$maFormule:=""
EDITER FORMULE([Salaires];$maFormule)
Si(OK=1)
   APPLIQUER A SELECTION([Salaires];EXECUTER FORMULE($maFormule))
Fin de si
```

Variables et ensembles système

Si l'utilisateur valide la boîte de dialogue, la variable système OK prend la valeur 1. Si l'utilisateur annule la boîte de dialogue, la variable système OK prend la valeur 0.

EXECUTER FORMULE

EXECUTER FORMULE (instruction)

Paramètre	Type		Description
instruction	Chaîne	⇨	Code à exécuter

Description

EXECUTER FORMULE exécute *instruction* comme une ligne de code. Cette chaîne d'instructions doit comporter une seule ligne. Si *instruction* est une chaîne vide, **EXECUTER FORMULE** ne fait rien.

Le principe est que si *instruction* peut être exécutée comme une méthode d'une seule ligne, alors elle s'exécutera correctement. La commande **EXECUTER FORMULE** doit être utilisée avec précautions, car elle ralentit la vitesse d'exécution. Dans une base compilée, le code d'*instruction* n'est pas compilé. Cela signifie que l'*instruction* sera bien exécutée, mais ne sera pas vérifiée par le compilateur au moment de la compilation.

Note : L'exécution de formules en mode compilé peut être optimisée à l'aide d'un cache (cf. paragraphe "Cache de formules en mode compilé" ci-dessous).

L'instruction peut notamment contenir les éléments suivants :

- un appel à une méthode projet,
- un appel à une commande 4D,
- une assignation.

La formule peut utiliser des variables process et interprocess. En revanche, *instruction* ne doit pas contenir d'instructions de contrôle de flux (**Si**, **Tant que**...) car le code doit "tenir" sur une seule ligne.

Pour assurer une évaluation correcte de l'*instruction* quelle que soit la langue ou la version de 4D, il est recommandé d'utiliser la syntaxe *tokenisée* pour les éléments dont le nom peut varier au fil des versions (commandes, tables, champs, constantes). Par exemple, pour insérer la commande **Heure courante**, saisissez '**Current time:C178**'. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser des tokens dans les formules**.

Cache de formules en mode compilé

A des fins d'optimisation, chaque formule exécutée via **EXECUTER FORMULE** en mode compilé peut être conservée en mémoire dans un cache dédié. La formule est stockée sous forme de références (*tokens*). Une fois placée dans le cache, une formule s'exécutera de manière beaucoup plus rapide par la suite car la phase de *tokenisation* sera évitée.

La taille du cache est de zéro par défaut (pas de cache) ; vous devez le créer et l'ajuster à l'aide de la commande **FIXER PARAMETRE BASE**. Par exemple :

```
FIXER PARAMETRE BASE(Nombre de formules en cache;0) //pas de cache de formules
FIXER PARAMETRE BASE(Nombre de formules en cache;3) //jusqu'à trois formules peuvent être en cache pour chaque process
```

La commande **EXECUTER FORMULE** utilise le cache uniquement lorsqu'elle est appelée depuis une base ou un composant exécuté(e) en mode compilé.

Exemple

Vous souhaitez exécuter des formules comportant des appels à des commandes 4D et à des tables. Ces éléments pouvant potentiellement être renommés, vous pouvez vous assurer d'une exécution correcte dans les versions futures de votre application en utilisant la syntaxe tokenisée :

```
EXECUTER FORMULE("Year of:C25 ([Produits:5]Date_Creation:2])+$ajout")
```

FIXER METHODES AUTORISEES

FIXER METHODES AUTORISEES (tabMéthodes)

Paramètre	Type		Description
tabMéthodes	Tableau chaîne	⇨	Tableau de noms de méthodes

Description

La commande **FIXER METHODES AUTORISEES** permet de définir les méthodes qui seront affichées dans l'éditeur de formules pour la session courante. Les méthodes désignées apparaîtront à la fin de la liste des commandes et pourront être utilisées dans les formules. Par défaut (si vous n'utilisez pas cette commande), aucune méthode n'est utilisable dans l'éditeur de formules. Si une formule utilise un nom de méthode non autorisée, une erreur de syntaxe est générée et la formule ne peut pas être validée.

Passez dans le paramètre *tabMéthodes* le nom d'un tableau contenant la liste de méthodes à proposer dans l'éditeur de formules. Le tableau doit avoir été défini préalablement.
Vous pouvez utiliser le caractère "joker" (@) dans les noms des méthodes afin de définir un ou plusieurs groupe(s) de méthodes autorisées.

Si vous souhaitez que l'utilisateur puisse appeler des commandes 4D non autorisées par défaut ou des commandes de plug-ins, vous devez utiliser des méthodes spécifiques chargées d'exécuter ces commandes.

Note : Le mécanisme de restriction d'accès aux commandes et méthodes dans l'éditeur de formules peut être désactivé pour tous les utilisateurs ou pour le Super_Utilisateur et l'Administrateur via une option des Propriétés de la base (page "Sécurité"). Si l'option "Désactivé pour tous" est sélectionnée, la commande **FIXER METHODES AUTORISEES** est sans effet.

Exemple

Cet exemple autorise toutes les méthodes dont le nom débute par "formule" et de la méthode "Total_général" dans l'éditeur de formules :

```
TABLEAU ALPHA(15;tabméthodes;2)
tabméthodes{1}:="formule@"
tabméthodes{2}:="Total_général"
FIXER METHODES AUTORISEES(tabméthodes)
```

LIRE METHODES AUTORISEES

LIRE METHODES AUTORISEES (tabMéthodes)

Paramètre	Type		Description
tabMéthodes	Tableau chaîne	⇦	Tableau de noms de méthodes

Description

La commande **LIRE METHODES AUTORISEES** retourne dans le tableau *tabMéthodes* le nom des méthodes “autorisées” dans l’éditeur de formules, c’est-à-dire pouvant être utilisées lors de l’écriture d’une formule — ces méthodes sont listées à la fin de la liste des commandes dans l’éditeur.

Par défaut, aucune méthode n’est utilisable dans l’éditeur de formules. Les méthodes doivent avoir été explicitement autorisées via la commande **FIXER METHODES AUTORISEES**. Si cette commande n’a pas été exécutée, **LIRE METHODES AUTORISEES** retourne une chaîne vide.

LIRE METHODES AUTORISEES retourne précisément ce qui a été passé à la commande **FIXER METHODES AUTORISEES**, c’est-à-dire un tableau alpha (la commande crée et dimensionne le tableau). En outre, si le caractère “joker” (@) a été utilisé pour désigner un groupe de méthodes, la chaîne contenant le caractère @ est retournée (et non les noms des méthodes du groupe).

Cette commande est utile pour préserver le paramétrage de l’ensemble courant de méthodes autorisées avant l’exécution d’une formule dans un contexte spécifique (par exemple un état rapide).

Exemple

Cet exemple permet d’autoriser ponctuellement un ensemble de méthodes spécifiques pour la création d’un état rapide :

```
 `Stockage du paramétrage courant
LIRE METHODES AUTORISEES(tabméthodes)

 `Définition des méthodes pour l'état
tabméthodes_Etats{1}:="Etats_@"
FIXER METHODES AUTORISEES(tabméthodes_Etats)
QR ETAT([Personnes];"MonEtat")

 `Rétablissement des paramètres courants
FIXER METHODES AUTORISEES(tabméthodes)
```

Gestion de la saisie

- ALLER A OBJET
- EDITER ELEMENT
- FILTRER FRAPPE CLAVIER
- Frappe clavier
- Lire texte edite
- SELECTIONNER TEXTE
- TEXTE SELECTIONNE

ALLER A OBJET

ALLER A OBJET ({* ;} objet)

Paramètre	Type		Description
*	Opérateur	⇨	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Champ, Variable	⇨	Nom d'objet (si * spécifié) sinon Variable ou champ saisissable à sélectionner

Description

La commande **ALLER A OBJET** permet de sélectionner l'objet saisissable *objet* (variable ou champ) en tant que zone active du formulaire. C'est l'équivalent d'un clic de l'utilisateur dans la zone ou de l'utilisation de la touche **Tabulation** pour sélectionner le champ ou la variable.

Si vous passez le paramètre optionnel *, vous indiquez que le second paramètre désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables texte uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Pour supprimer tout focus dans le formulaire courant, appelez la commande en passant un nom d'objet vide dans *objet* (cf. exemple 2).

La commande **ALLER A OBJET** peut être utilisée dans le contexte d'un sous-formulaire. Lorsqu'elle est appelée depuis un sous-formulaire, elle recherche en premier lieu objet dans le sous-formulaire puis, si la recherche n'aboutit pas, elle étend la recherche aux objets du formulaire parent.

Exemple 1

Voici les deux modes d'utilisation de la commande **ALLER A OBJET** :

```
ALLER A OBJET([Personnel]Nom) `Référence de champ
ALLER A OBJET(*;"ZonePrénoms") `Nom d'objet
```

Exemple 2

Vous souhaitez que plus aucun objet du formulaire n'ait le focus :

```
ALLER A OBJET(*;"")
```

Exemple 3

Reportez-vous à l'exemple de la commande **REFUSER**.

EDITER ELEMENT

EDITER ELEMENT ({* ;} objet {; élément})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un table ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Table ou variable (si * omis)
élément	Entier long	⇒	Numéro d'élément

Description

La commande **EDITER ELEMENT** permet de passer en "mode édition" l'élément courant ou l'élément de numéro *élément* du tableau ou de la liste désigné(e) par le paramètre *objet*.
Le mode édition signifie que l'élément est sélectionné et prêt à être modifié : la saisie d'un caractère remplacera intégralement le contenu de l'élément.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (dans ce cas, passez une chaîne dans *objet*). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est une table ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de table ou une variable.

Cette commande s'applique aux objets saisissables suivants :

- Listes hiérarchiques
- List box
- Sous-formulaires (dans ce cas, seul un nom d'objet — le sous-formulaire — peut être passé dans *objet*),
- Formulaires liste affichés via la commande **VISUALISER SELECTION** ou **MODIFIER SELECTION**.

Si la commande est utilisée avec un objet saisissable qui n'est pas une liste, elle équivaut à la commande **ALLER A OBJET**.
La commande ne fait rien si la liste ou le tableau désigné(e) est vide ou invisible. Si la liste ou le tableau n'est pas saisissable, la commande sélectionne (sans passer en édition) l'élément spécifié.
Dans le cadre d'une list box, si la colonne n'autorise pas la saisie de texte (saisie par case à cocher ou menu déroulant uniquement), l'élément spécifié prend le focus.

Le paramètre facultatif *élément* vous permet de désigner la position de l'élément (liste hiérarchique) ou le numéro de la ligne (list box, formulaire liste et sous-formulaire en mode "multi-sélection") à passer en édition. Si vous ne passez pas ce paramètre, la commande s'applique à l'élément courant de l'*objet*. S'il n'y a pas d'élément courant, le premier élément de l'*objet* passe en édition.

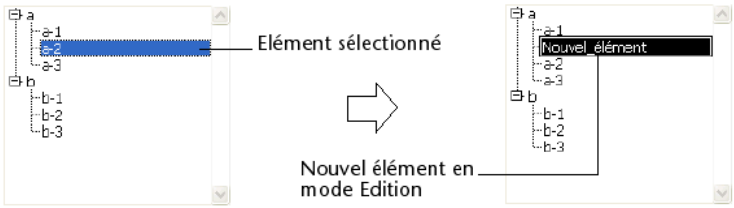
Notes :

- Dans le cadre des sous-formulaires et des formulaires liste, la commande passe en édition le premier champ de la ligne spécifiée, dans l'ordre de saisie.
- Dans le cadre d'une list box affichée en mode hiérarchique, si l'élément visé appartient à un niveau hiérarchique contracté, le niveau ainsi que les éventuels niveaux parents sont automatiquement déployés afin que la ligne soit visible.

Exemple 1

Cette commande peut être utile notamment lors de la création d'un nouvel élément de liste hiérarchique. Au moment de l'appel de la commande, le dernier élément ajouté ou inséré dans la liste devient automatiquement éditable, sans que l'utilisateur n'ait à effectuer d'action spécifique.
Le code suivant pourrait être la méthode d'un bouton permettant d'insérer un nouvel élément dans une liste existante. Le libellé "Nouvel_élément" proposé par défaut est automatiquement placé en mode édition :

```
vlUniqueRef:=vlUniqueRef+1
INSERER DANS LISTE(hList;*;"Nouvel_élément";vlUniqueRef)
EDITER ELEMENT(*;"MaListe")
```



Exemple 2

Soient deux colonnes d'une list box dont les noms de variables associées sont respectivement "Tableau1" et "Tableau2". L'exemple suivant insère un nouvel élément dans les deux tableaux et passe le nouvel élément du tableau 2 en mode édition :

```
$vlNumLigne:=Taille tableau(Tableau1)+1
LISTBOX INSERER LIGNES(*;"MaListBox";$vlNumLigne)
Tableau1{$vlNumLigne}:="Nouvelle valeur 1"
Tableau2{$vlNumLigne}:="Nouvelle valeur 2"
EDITER ELEMENT(Tableau2;$vlNumLigne)
```

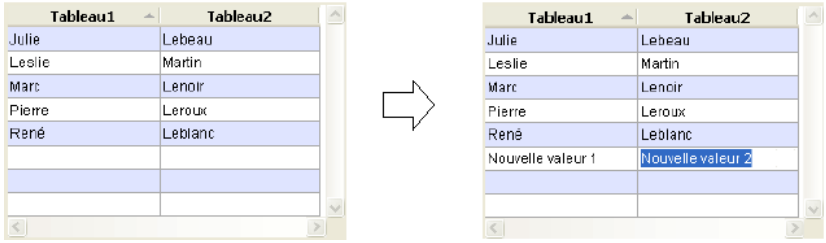


Tableau1
Tableau2
Julie
Lebeau
Leslie
Martin
Marc
Lenoir
Pierre
Leroux
René
Leblanc
Tableau1
Tableau2
Julie
Lebeau
Leslie
Martin
Marc
Lenoir
Pierre
Leroux
René
Leblanc
Nouvelle valeur 1
Nouvelle valeur 2

FILTRER FRAPPE CLAVIER

FILTRER FRAPPE CLAVIER (carFiltré)

Paramètre	Type		Description
carFiltré	Chaîne	⇨	Caractère(s) de remplacement ou Chaîne vide pour annuler le filtrage clavier

Description

FILTRER FRAPPE CLAVIER vous permet de remplacer le caractère saisi par l'utilisateur dans un champ ou une zone saisissable par le premier caractère de la chaîne *carFiltré*.

Si vous passez une chaîne vide, le filtrage clavier en cours est annulé.

Vous appelez généralement **FILTRER FRAPPE CLAVIER** dans une méthode formulaire ou objet lorsque vous gérez l'événement formulaire Sur avant frappe clavier. Pour détecter les événements de frappe clavier, utilisez la commande **Evenement formulaire**. Pour récupérer les caractères saisis au clavier, utilisez les fonctions **Frappe clavier** ou **Lire texte edite**.

IMPORTANT : Si vous voulez effectuer des opérations “à la volée” en fonction de la valeur courante de la zone saisissable en cours de modification ainsi que du caractère à saisir, rappelez-vous que le texte affiché à l'écran n'est pas encore la valeur du champ ou de la variable. La valeur saisie dans une variable ou un champ ne lui est affectée que lorsque la zone est validée (lorsque l'utilisateur appuie sur la touche **Tabulation**, clique sur un bouton, etc.). En conséquence, pensez à placer les valeurs saisies dans une variable temporaire et à travailler avec celle-ci, puis à assigner cette variable à la zone de saisie (reportez-vous à l'exemple ci-dessous). Vous pouvez également utiliser la fonction **Lire texte edite**.

Utilisez la commande **FILTRER FRAPPE CLAVIER** dans les cas suivants :

- Pour effectuer un filtrage personnalisé des caractères,
- Pour créer un filtre de saisie non disponible en standard,
- Pour implémenter des zones de recherche ou de pré-saisie dynamiques.

ATTENTION : si vous appelez la commande **Frappe clavier** après avoir appelé **FILTRER FRAPPE CLAVIER**, c'est le caractère passé à cette commande qui sera retourné et non le caractère réellement saisi.

Exemple 1

Avec le code suivant :

```
   ` Méthode objet de la zone saisissable monObjet
Au cas ou
   :(Evenement formulaire=Sur chargement)
      monObjet:=""
   :(Evenement formulaire=Sur avant frappe clavier)
      Si(Position(Frappe clavier;"0123456789")>0)
         FILTRER FRAPPE CLAVIER("*")
      Fin de si
Fin de cas
```

... tous les chiffres saisis dans la zone *monObjet* seront transformés en astérisques.

Exemple 2

Le code ci-dessous définit le comportement d'une zone de saisie de mot de passe, dans laquelle les caractères saisis sont remplacés à l'écran par des caractères aléatoires :

```
   ` Méthode objet de la zone saisissable vaMotsPasse
Au cas ou
   :(Evenement formulaire=Sur chargement)
      vaMotsPasse:=""
      vaMotPasseActuel:=""
   :(Evenement formulaire=Sur avant frappe clavier)
      Gérer frappe clavier(->vaMotsPasse;->vaMotPasseRéel)
      Si(Position(Frappe clavier;Caractere(Touche retour arrière)+Caractere(Touche gauche)+
         Caractere(Touche droite)+Caractere(Touche haut)+
         Caractere(Touche bas))=0)
         FILTRER FRAPPE CLAVIER(Caractere(65+(Hasard%26)))
      Fin de si
Fin de cas
```

Une fois la zone validée, vous récupérez le mot de passe réellement saisi par l'utilisateur dans la variable *vaMotPasseRéel*. La méthode ***Gérer frappe clavier*** est listée dans l'exemple de la commande **Frappe clavier**.

Exemple 3

Vous disposez dans votre application de diverses zones de texte dans lesquelles vous pouvez saisir quelques phrases. Votre application comporte également une table de glossaire contenant les termes les plus fréquemment utilisés dans votre base. Lors de l'édition de vos zones de texte, vous voulez pouvoir rapidement, à partir du glossaire, retrouver et insérer des mots en fonction des caractères sélectionnés dans le texte. Pour cela, vous avez deux solutions : soit placer des boutons avec des touches associées qui vont exécuter l'opération, soit intercepter les frappes clavier spéciales pendant la saisie. L'exemple ci-dessous utilise la seconde solution, basée sur la touche **Aide**.

Comme décrit ci-dessus, lorsque vous éditez une zone de texte, la valeur du champ ou de la variable de texte ne sera réellement modifiée que lorsque que vous l'aurez validée. Pour retrouver et insérer rapidement des entrées du glossaire dans une zone de texte alors qu'elle est en train d'être modifiée, vous devez donc créer une seconde zone "tampon" pour y placer les valeurs saisies. Vous pouvez effectuer cette opération à l'aide de la méthode projet décrite ci-dessous. Vous passez comme premiers paramètres des pointeurs vers la zone de saisie et vers la variable, puis la chaîne de caractère “interdits” comme troisième paramètre. Peu importe comment l'entrée clavier sera traitée, la méthode retourne la valeur saisie originale. Les caractères “interdits” sont les caractères que vous ne voulez pas insérer dans la zone saisissable et que vous voulez traiter en tant que caractères spéciaux.

```
  ` Méthode projet Frappe clavier tampon
  ` Frappe clavier tampon ( Pointeur ; Pointeur ; Alpha ) -> Alpha
  ` Frappe clavier tampon ( -> zoneSource ; -> valeurCourante ; Filtre ) -> Ancien frappe clavier
C_ALPHA(1;$0)
C_POINTEUR($1;$2)
C_TEXTE($vtNouvValeur)
C_ALPHA(255;$3)
  ` Retourne la frappe clavier originale
$0:=Frappe clavier
  ` Obtenir la sélection de texte dans la zone saisissable
TEXTE SELECTIONNE($1->;$vlDébut;$vlFin)
  ` Commencer à travailler sur la valeur courante
$vtNouvValeur:=$2->
  ` En fonction de la touche enfoncée ou du caractère saisi, effectuer les actions appropriées
Au cas ou
  ` La touche Retour arrière a été enfoncée
   :(Code de caractere($0)=Touche retour arrière)
  ` Supprimer les caractères sélectionnés ou le caractère à gauche du curseur
      $vtNouvValeur:=Supprimer texte($vtNouvValeur;$vlDébut;$vlFin)
  ` Une touche "flèche" a été appuyée
  ` Ne faites rien sauf accepter la frappe clavier
   :(Code de caractere($0)=Touche gauche)
   :(Code de caractere($0)=Touche droite)
   :(Code de caractere($0)=Touche haut)
   :(Code de caractere($0)=Touche bas)

  ` Un caractère valide a été saisi
   :(Position($0;$3)=0)
      $vtNouvValeur:=Inserer texte($vtNouvValeur;$vlDébut;$vlFin;$0)
   Sinon
  ` Le caractère n'est pas accepté
      FILTRER FRAPPE CLAVIER("")
Fin de cas
  ` Retourner la valeur pour la prochaine gestion de la frappe clavier
$2->:=$vtNouvValeur
```

Cette méthode utilise les sous-méthodes suivantes :

```
  ` Méthode projet Supprimer texte
  ` Supprimer texte ( Alpha ; Long ; Long ) -> Alpha
  ` Supprimer texte ( -> Texte ; SelDébut ; SelFin ) -> Nouveau texte
C_TEXTE($0;$1)
C_ENTIER LONG($2;$3)
$0:=Sous chaine($1;1;$2-1-Num($2=$3))+Sous chaine($1;$3)
```

```
  ` Méthode projet Inserer texte
  ` Inserer texte ( Alpha ; Long ; Long ; Alpha ) -> Alpha
  ` Inserer texte ( -> texteSource ; SelDébut ; SelFin ; Texte à insérer ) -> Nouveau texte
C_TEXTE($0;$1;$4)
C_ENTIER LONG($2;$3)
$0:=$1
Si($2#$3)
   $0:=Sous chaine($0;1;$2-1)+$4+Sous chaine($0;$3)
Sinon
   Au cas ou
      :($2<=1)
         $0:=$4+$0
      :($2>Longueur($0))
         $0:=$0+$4
      Sinon
         $0:=Sous chaine($0;1;$2-1)+$4+Sous chaine($0;$2)
   Fin de cas
Fin de si
```

Une fois que vous avez ajouté ces méthodes projet à votre base, vous pouvez les utiliser de la manière suivante :

```
  ` Méthode objet de la zone saisissable vaDescription
Au cas ou
   :(Evenement formulaire=Sur chargement)
      vaDescription:=""
      vaDescriptionDouble:=""
  ` Etablir la liste des caractères “interdits” à traiter comme des touches spéciales
  ` (Dans cet exemple, seule la touche Aide est filtrée)
```

```
      vaTouchesSpéciales:=Caractere(Touche aide)
   :(Evenement formulaire=Sur avant frappe clavier)
      $vsKey:=Frappe clavier tampon(->vaDescription;->vaDescriptionDouble;vaTouchesSpéciales)
      Au cas ou
         :(Code de caractere($vsKey)=Touche aide)
   ` Faire quelque chose lorsque la touche Aide est enfoncée
   ` Dans cet exemple, une saisie de glossaire doit être recherchée et insérée
            chercher_Glossaire(->vaDescription;->vaDescriptionDouble)
      Fin de cas
Fin de cas
```

La méthode projet ***chercher_Glossaire*** est listée ci-dessous (le point principal est l'utilisation de la variable tampon pour réaffecter la zone saisissable à modifier) :

```
   ` Méthode projet chercher_Glossaire
   ` chercher_Glossaire ( Pointeur ; Pointeur )
   ` chercher_Glossaire ( -> zone saisissable ; ->variable double )
C_POINTEUR($1;$2)
C_ENTIER LONG($vlDébut;$vlFin)
   ` Obtenir la sélection de texte dans la zone saisissable
TEXTE SELECTIONNE($1->;$vlDébut;$vlFin)
   ` Obtenir le texte sélectionné ou le mot situé à gauche du curseur $vtTexteSelectionne:=obtenirTexteSelectionne
($2->;$vlDébut;$vlFin)
   ` Y a-t-il quelque chose à rechercher ?
Si($vtTexteSelectionne#"")
   ` Si la sélection de texte était le curseur, la sélection débute au mot situé après le curseur
   Si($vlDébut=$vlFin)
      $vlDébut:=$vlDébut-Longueur($vtTexteSelectionne)
   Fin de si
   ` Chercher la première entrée du glossaire disponible
   CHERCHER([Glossaire];[Glossaire]Saisie=$vtTexteSelectionne+"@")
   ` Existe-t-elle ?
   Si(Enregistrements trouves([Glossaire])>0)
   ` Si oui, l'insérer dans la zone tampon
      $2->:=Inserer texte($2->;$vlDébut;$vlFin;[Glossaire]Saisie)
   ` Copier le tampon dans la zone saisissable
      $1->:=$2->
   ` Fixer la sélection après avoir inséré l'entrée du glossaire
      $vlFin:=$vlDébut+Longueur([Dictionnaire]Saisie)
      SELECTIONNER TEXTE(vsComments;$vlFin;$vlFin)
   Sinon
   ` Il n'y a pas d'entrée qui correspond dans le glossaire
      BEEP
   Fin de si
Sinon
   ` Il n'y a pas de texte sélectionné
   BEEP
Fin de si
```

La méthode ***obtenirTexteSelectionne*** est la suivante :

```
   ` Méthode objet obtenirTexteSelectionne
   ` obtenirTexteSelectionne ( Alpha ; Entier long ; Entier long ) -> Alpha
   ` obtenirTexteSelectionne ( Texte ; SelDébut ; SelFin ) -> texte sélectionné
C_TEXTE($0;$1)
C_ENTIER LONG($2;$3)
Si($2<$3)
   $0:=Sous chaine($1;$2;$3-$2)
Sinon
   $0:=""
   $2:=$2-1
   Repeter
      Si($2>0)
         Si(Position($1≤$2≥;"  ,.!?:;()-_--")=0)
            $0:=$1≤$2≥+$0
            $2:=$2-1
         Sinon
            $2:=0
         Fin de si
      Fin de si
   Jusque($2=0)
Fin de si
```

Frappe clavier

Frappe clavier -> Résultat

Paramètre	Type	Description
Résultat	Chaîne	Caractère saisi par l'utilisateur

Description

Frappe clavier retourne le caractère tapé par l'utilisateur dans un champ ou une zone saisissable.

En général, vous appelez **Frappe clavier** dans une méthode formulaire ou objet, lors de la gestion des événements formulaire Sur avant frappe clavier et Sur après frappe clavier. Pour détecter les événements de frappe clavier, utilisez la commande **Evenement formulaire**.

Si vous voulez remplacer un caractère saisi par l'utilisateur par un autre, utilisez la commande **FILTRER FRAPPE CLAVIER**.

IMPORTANT : Si vous voulez effectuer des opérations “à la volée” en fonction de la valeur courante de la zone saisissable en cours de modification ainsi que du caractère à saisir, rappelez-vous que le texte affiché à l'écran n'est pas encore la valeur du champ ou de la variable. La valeur saisie dans une variable ou un champ ne lui est affectée que lorsque la zone est validée (si l'utilisateur appuie sur la touche Tabulation, clique sur un bouton, etc.). En conséquence, pensez à placer les valeurs saisies dans une variable temporaire et à travailler avec celle-ci, ou utilisez la commande **Lire texte edite**. Vous devez procéder ainsi si vous souhaitez connaître la valeur courante du texte pour effectuer des actions spéciales.

Vous pouvez utiliser la commande **Frappe clavier** pour :

- effectuer un filtrage personnalisé des caractères
- créer un filtre de saisie non disponible en standard, par exemple dans les filtres de saisie
- implémenter des zones de recherche ou de pré-saisie dynamiques.

Note : Vous ne pouvez pas utiliser la fonction **Frappe clavier** dans les sous-formulaires.

Exemple 1

Référez-vous aux exemples de la commande **FILTRER FRAPPE CLAVIER**.

Exemple 2

Lorsque vous traitez un événement Sur avant frappe clavier, vous gérez la modification de la zone de texte courante (celle qui contient le curseur), et non la “valeur future” de la source de données (champ ou variable) de cette zone. La méthode **Gérer frappe clavier** décrite ci-dessous vous permet de placer dans une seconde variable les caractères saisis dans une zone de texte. Vous pouvez alors utiliser cette variable pour effectuer différentes actions pendant la saisie des caractères dans la zone. Vous passez comme premier paramètre un pointeur vers la source des données de la zone, et comme second paramètre un pointeur vers cette seconde variable. La méthode renvoie la nouvelle valeur de la zone de texte dans la seconde variable et retourne **Vrai** si cette valeur est différente de ce qu'elle était avant la saisie du dernier caractère.

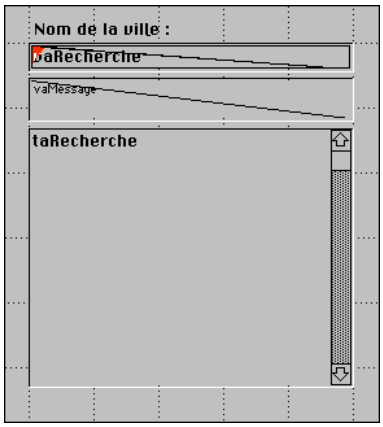
```
  ` Méthode projet Gérer frappe clavier
  ` Gérer frappe clavier ( Pointeur ; Pointeur ) -> Booléen
  ` Gérer frappe clavier ( -> zoneSource ; -> valeurCourante ) -> Est-ce une nouvelle valeur

C_POINTEUR($1;$2)
C_TEXTE($vtNouvValeur)

  ` Récupérer le texte sélectionné dans la zone saisissable
TEXTE SELECTIONNE($1->;$vlDébut;$vlFin)
  ` Commencer à travailler avec la valeur courante
$vtNouvValeur:=$2->
  ` Selon la touche appuyée ou le caractère saisi, effectuer les actions appropriées
Au cas ou

  ` La touche Retour arrière a été enfoncée
  :(Code de caractere(Frappe clavier)=Touche retour arrière)
  ` Supprimer les caractères sélectionnés ou le caractère à gauche du curseur
     $vtNouvValeur:=Sous chaine($vtNouvValeur;1;$vlDébut-1-Num($vlDébut=$vlFin))
     +Sous chaine($vtNouvValeur;$vlFin)

  ` Un caractère acceptable a été saisi
  :(Position(Frappe clavier;"abcdefghjiklmnopqrstuvwxyz -0123456789")>0)
     Si($vlDébut#$vlFin)
  ` Un ou plusieurs caractères sont sélectionnés, la frappe clavier va les effacer
        $vtNouvValeur:=Sous chaine($vtNouvValeur;1;$vlDébut-1)+Frappe clavier+Sous chaine($vtNouvValeur;$vlFin)
     Sinon
  ` La sélection de texte est le curseur
        Au cas ou
  ` Le curseur est actuellement au début du texte
           :($vlDébut<=1)
  ` Insertion du caractère au début du texte
              $vtNouvValeur:=Frappe clavier+$vtNouvValeur
  ` Le curseur est actuellement à la fin du texte
           :($vlDébut>=Longueur($vtNouvValeur))
  ` Ajouter le caractère à la fin du texte
              $vtNouvValeur:=$vtNouvValeur+Frappe clavier
```

```
            Sinon
   ` Le curseur se trouve dans le texte, insérer le nouveau caractère
               $vtNouvValeur:=Sous chaine($vtNouvValeur;1;$vlDébut-1)+Frappe clavier
               +Sous chaine($vtNouvValeur;$vlDébut)
            Fin de cas
         Fin de si

   ` Une touche flèche a été enfoncée
   ` Ne rien faire, mais valider la frappe clavier
      :(Code de caractere(Frappe clavier)=Touche gauche)
      :(Code de caractere(Frappe clavier)=Touche droite)
      :(Code de caractere(Frappe clavier)=Touche haut)
      :(Code de caractere(Frappe clavier)=Touche bas)
   `
      Sinon
   ` Il ne faut pas accepter des caractères autres que des lettres, chiffres, espaces et tirets
         FILTRER FRAPPE CLAVIER("")
   Fin de cas
    ` Est-ce que la valeur est maintenant différente ?
   $0:=($vtNouvValeur#$2->)
    ` Retourner la valeur pour la gestion de la prochaine frappe clavier
   $2->:=$vtNouvValeur
```

Une fois que vous avez ajouté cette méthode projet à votre application, vous pouvez l'utiliser ainsi :

```
    ` Méthode objet de la zone saisissable MonObjet
   Au cas ou
      :(Evenement formulaire=Sur chargement)
         MonObjet:=""
         MonObjetCaché:=""
      :(Evenement formulaire=Sur avant frappe clavier)
         Si(Gérer frappe clavier(->MonObjet;->MonObjetCaché))
    ` Effectuer des actions appropriées par rapport à la valeur stockée dans MonObjetCaché
         Fin de si
   Fin de cas
```

Examinons par exemple le formulaire suivant :

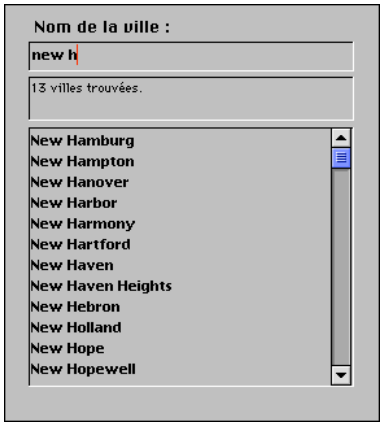


Il est composé des objets suivants : une zone saisissable *vaRecherche*, une zone non-saisissable *vaMessage* et une zone de défilement *taRecherche*. Lorsque l'utilisateur saisit des caractères dans *vaRecherche*, la méthode objet effectue une recherche sur la table [Codes postaux] permettant d'afficher des villes américaines en saisissant seulement les premiers caractères de leur nom. Voici la méthode objet de *vaRecherche* :

```
    ` Méthode objet de la zone saisissable vaRecherche
   Au cas ou
      :(Evenement formulaire=Sur chargement)
         vaRecherche:=""
         vaRésultat:=""
         vaMessage:="Saisissez les premiers caractères de la ville que vous cherchez."
         EFFACER VARIABLE(taRecherche)
      :(Evenement formulaire=Sur avant frappe clavier)
         Si(Gérer frappe clavier(->vaRecherche;->vaRésultat))
            Si(vaRésultat#"")
               CHERCHER([Codes postaux];[Codes postaux]Ville=vaRésultat+"@")
               SUPPRIMER MESSAGES
               VALEURS DISTINCTES([Codes postaux]Ville;taRecherche)
               LAISSER MESSAGES
               $vlRésultat:=Taille tableau(taRecherche)
               Au cas ou
                  :($vlRésultat=0)
                     vaMessage:="Aucune ville trouvée."
                  :($vlRésultat=1)
                     vaMessage:="Une ville trouvée."
                  Sinon
                     vaMessage:=Chaine($vlRésultat)+" villes trouvées."
               Fin de cas
```

```
            Sinon
               SUPPRIMER DANS TABLEAU(taRecherche;1;Taille tableau(taRecherche))
               vaMessage:="Saisissez les premières lettres de la ville que vous cherchez."
            Fin de si
         Fin de si
   Fin de cas
```

Voici le formulaire en exécution :



A l'aide des possibilités de communication interprocess de 4D, vous pouvez construire une interface dans laquelle les recherches se construisent dans des palettes flottantes communiquant avec les process dans lesquels les enregistrements sont affichés ou modifiés.

Lire texte edite

Lire texte edite -> Résultat

Paramètre	Type	Description
Résultat	Texte	Texte en cours de saisie

Description

La commande **Lire texte edite** retourne le texte en cours de saisie dans un objet de formulaire.

Cette commande est principalement destinée à être utilisée avec l'événement formulaire Sur après frappe clavier pour récupérer le texte au fur et à mesure de la frappe. Elle peut également être utilisée avec l'événement formulaire Sur avant frappe clavier.

La combinaison de cette commande avec les événements formulaire Sur avant frappe clavier et Sur après frappe clavier fonctionne de la manière suivante :

- Dès qu'un caractère est tapé au clavier, l'événement Sur avant frappe clavier est généré. Dans cet événement, la fonction **Lire texte edite** retourne le contenu de la zone avant la dernière frappe clavier. Par exemple, si la zone contient "PA" et que l'utilisateur tape "R", **Lire texte edite** retourne "PA" dans l'événement Sur avant frappe clavier. Si la zone ne contient rien au départ, **Lire texte edite** retourne une chaîne vide.
- Ensuite, l'événement formulaire Sur après frappe clavier est généré. Dans cet événement, la fonction **Lire texte edite** retourne le contenu de la zone y compris le dernier caractère entré au clavier. Par exemple, si la zone contient "PA" et que l'utilisateur tape "R", **Lire texte edite** retourne "PAR" dans l'événement Sur après frappe clavier.

Ces deux événements ne sont générés que dans les méthodes des objets concernés.

Dans un contexte autre que la saisie dans un formulaire, cette fonction retourne une chaîne vide.

Exemple 1

Dans un formulaire entrée, vous souhaitez que les caractères saisis soient automatiquement mis en majuscules :

```
Si(Evenement formulaire=Sur après frappe clavier)
   [Voyages]Agences:=Majusc(Lire texte edite)
Fin de si
```

Exemple 2

Voici un exemple de traitement à la volée des caractères saisis dans un champ texte. Le principe consiste à placer dans un autre champ texte (appelé "Mots") la décomposition en mots de la phrase en cours de saisie. Pour cela, écrivez dans la méthode objet du champ de saisie :

```
Si(Evenement formulaire=Sur après frappe clavier)
   $SaisieTempsRéel:=Lire texte edite
   PROPRIETES PLATE FORME($plate_forme)
   Si($plate_forme#3) ` Macintosh ou Power Macintosh
      Repeter
         $PhraseDécomposée:=Remplacer chaine($SaisieTempsRéel;Caractere(32);Caractere(13))
      Jusque(Position(" ";$PhraseDécomposée)=0)
   Sinon ` Windows
      Repeter
         $PhraseDécomposée:=Remplacer chaine($SaisieTempsRéel;Caractere(32);
         Caractere(13)+Caractere(10))
      Jusque(Position(" ";$PhraseDécomposée)=0)
   Fin de si
   [Exemple]Mots:=$PhraseDécomposée
Fin de si
```

Note : Cet exemple n'est pas exhaustif puisque l'on considère que les mots sont séparés par des espaces uniquement (**Caractere (32)**). La mise au point d'un système complet nécessiterait l'ajout d'autres filtres afin de repérer tous les mots (point-virgules, virgules, apostrophes, etc...).

SELECTIONNER TEXTE

SELECTIONNER TEXTE ({* ;} objet ; débutSél ; finSél)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Champ, Variable, Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable saisissable (si * est omis)
débutSél	Entier long	⇒	Nouvelle position de début de sélection de texte
finSél	Entier long	⇒	Nouvelle position de fin de sélection de texte

Description

La commande **SELECTIONNER TEXTE** sélectionne une partie du texte dans *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre *objet* désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables de formulaire uniquement).

Si *objet* n'est pas l'objet en cours de modification, il récupère le focus.

Note : Cette commande ne peut pas être utilisée avec des champs situés dans un sous-formulaire.

Le paramètre *débutSél* représente la position du premier caractère à sélectionner, et le paramètre *finSél* représente la position du dernier caractère à sélectionner plus un. Si *débutSél* et *finSél* sont identiques, le point d'insertion est placé devant le caractère spécifié par *débutSél* et aucun caractère n'est sélectionné.

Si *finSél* est supérieur au nombre de caractères présents dans l'objet, tous les caractères compris entre *débutSél* et la fin du texte sont sélectionnés.

Exemple 1

L'exemple suivant sélectionne tous les caractères dans le champ saisissable *[Produits]Notes* :

```
SELECTIONNER TEXTE([Produits]Notes;1;Longueur([Produits]Notes)+1)
```

Exemple 2

L'exemple suivant place le point d'insertion au début du champ *[Produits]Notes* :

```
SELECTIONNER TEXTE([Produits]Notes;1;1)
```

Exemple 3

L'exemple suivant place le point d'insertion à la fin du champ *[Produits]Notes* :

```
$vLen:=Longueur([Produits]Notes)+1
SELECTIONNER TEXTE([Produits]Notes;$vLen;$vLen)
```

Exemple 4

Reportez-vous à l'exemple de la commande **FILTRER FRAPPE CLAVIER**.

TEXTE SELECTIONNE

TEXTE SELECTIONNE ({* ;} objet ; débutSél ; finSél)

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Champ, Variable, Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
débutSél	Entier long	⬅	Position du début de la sélection de texte
finSél	Entier long	⬅	Position de la fin de la sélection de texte

Description

La commande **TEXTE SELECTIONNE** vous permet de déterminer précisément le texte actuellement sélectionné dans *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre *objet* désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables de formulaire uniquement).

Note : Cette commande ne peut pas être utilisée avec des champs situés dans le formulaire en mode liste d'un sous-formulaire.

Le texte peut être sélectionné par l'utilisateur ou par la commande **SELECTIONNER TEXTE**.

Le paramètre *débutSél* retourne la position du premier caractère sélectionné.
Le paramètre *finSél* retourne la position du dernier caractère sélectionné plus un.

Si les valeurs *débutSél* et *finSél* retournées sont identiques, l'utilisateur n'a pas sélectionné de texte et le point d'insertion est placé devant le caractère spécifié par *débutSél*.

Si l'objet désigné par le paramètre *objet* n'est pas trouvé dans le formulaire, la commande retourne -1 dans *débutSél* et -2 dans *finSél*.

Exemple 1

L'exemple suivant récupère le texte sélectionné dans le champ *[Produits]Notes* :

```
TEXTE SELECTIONNE([Produits]Notes;vPremier;vDernier)
Si(vPremier<vDernier)
   ALERTE("Le texte sélectionné est : "+Sous chaine([Produits]Notes;vPremier;vDernier-vPremier))
Fin de si
```

Exemple 2

Reportez-vous à l'exemple de la commande **FILTRER FRAPPE CLAVIER**.

Exemple 3

Modification du style du texte sélectionné :

```
TEXTE SELECTIONNE(*;"monTexte";$debutsel,$finsel)
ST FIXER ATTRIBUTS(*;"monTexte";$debutsel,$finsel;Attribut style souligné;1;Attribut style gras;1)
```

Gestion du cache

4D utilise en interne un système intégré de cache de données permettant d'accélérer les opérations d'E/S. Le fait que des modifications de données soient, par moments, présentes dans le cache de données et pas sur le disque est entièrement transparent pour votre code. Par exemple, si vous appelez la commande **CHERCHER**, le moteur de 4D va intégrer les données présentes dans le cache pour effectuer l'opération.

Le gestionnaire du cache de la base de données a été complètement réécrit en 4D v16, permettant ainsi de tirer parti des environnements 64 bits. Automatiquement disponible et optimisé, il peut cependant être configuré ou analysé dynamiquement avec les commandes de ce thème.

Notez aussi que le sélecteur Cache flush periodicity peut être utilisé avec les commandes **FIXER PARAMETRE BASE** et **Lire parametre base**.

- ECRIRE CACHE Modifié 16.0
- FIXER TAILLE CACHE Nouveauté 16.0
- Lire informations cache Nouveauté 16.0
- LIRE STATISTIQUES MEMOIRE Modifié 16.0
- Lire taille cache Nouveauté 16.0

ECRIRE CACHE

ECRIRE CACHE {(taille | *)}

Paramètre	Type		Description
taille | *	Réel, Opérateur	⇨	* pour vider le cache, ou nombre d'octets minimum de libération du cache

Description

La commande **ECRIRE CACHE** sauvegarde immédiatement le cache de données sur le disque. Toutes les modifications apportées à la base sont alors stockées sur disque.

Par défaut, cette commande n'affecte pas le contenu courant du cache, ce qui signifie que les données qu'il contient restent utilisables lors des accès en lecture ultérieurs. Optionnellement, vous pouvez passer un paramètre pour le modifier :

- passez * pour sauvegarder le cache et vider entièrement le cache de la mémoire,
- passez une valeur pour sauvegarder le cache et libérer au minimum le nombre *taille* d'octets dans le cache.

Note : Passer un paramètre à cette commande est à envisager uniquement pour effectuer des tests. Pour des raisons de performances, il est fortement déconseillé de vider le cache en environnement de production.

En temps normal, vous n'avez pas à appeler cette commande, car 4D sauvegarde régulièrement les modifications. Il est préférable d'utiliser l'option **Ecriture cache toutes les X mn/secondes** (option de la page **Page Base de données/Mémoire** des Propriétés de la base), qui spécifie les intervalles de sauvegarde des données, afin de contrôler l'écriture du cache de données sur le disque. Il est recommandé d'utiliser la valeur par défaut, qui est de 20 secondes. Notez également que le paramètre Périodicité écriture cache peut être utilisé avec les commandes **FIXER PARAMETRE BASE** et **Lire parametre base** pour fixer ou lire cet intervalle.

FIXER TAILLE CACHE

FIXER TAILLE CACHE (taille {; libereMini})

Paramètre	Type		Description
taille	Réel	➡	Taille du cache du la base de données en octets
libereMini	Réel	➡	Nombre minimum d'octets à libérer lorsque le cache est plein

64 bits

Cette commande fonctionne uniquement dans les versions 64 bits de 4D.

Description

La commande **FIXER TAILLE CACHE** fixe dynamiquement la taille du cache de la base de données et, optionnellement, permet de fixer la taille minimum en octets à partir de laquelle on commence à libérer la mémoire..

Note : cette commande fonctionne uniquement en mode local (4D Server et 4D); elle ne doit pas être utilisée à partir d'un 4D en accès distant.

Dans *taille*, passez la nouvelle taille du cache de la base en octets. Cette nouvelle taille s'applique dynamiquement, dès que la commande est exécutée.

Dans *libereMini*, passez la taille minimum de mémoire à libérer dans le cache de la base de données, lorsque le moteur a besoin de plus d'espace pour allouer un objet en mémoire (valeur en octets). L'intérêt de cette option est de réduire le nombre de fois où les données sont libérées à partir de la mémoire cache afin d'obtenir de meilleures performances. Par défaut, si cette option n'est pas utilisée, 4D décharge au moins 10 % de la mémoire cache lorsqu'il a besoin de place. Si votre base de données fonctionne avec un grand cache, il peut être avantageux d'utiliser une taille fixe, qui ne dépend pas de la taille du cache. Vous pouvez régler ce paramètre en fonction de la taille des blocs de données traitées dans votre base de données.

Exemple

Vous voulez ajouter 100 Mo à la taille du cache de votre base. Vous pouvez écrire :

```
C_REEL($currentCache)
$currentCache:=Lire taille cache
 // la taille actuelle du cache est par exemple, 419430400
FIXER TAILLE CACHE($currentCache+100000000)
 // la taille courante du cache est maintenant de 519430400
```

Lire informations cache

Lire informations cache {(dbFilter)} -> Résultat

Paramètre	Type		Description
dbFilter	Objet	➡	définit la liste des attributs à retourner (filtrés par DB)
Résultat	Objet	⮌	Informations à propos du cache

64 bits

Cette commande fonctionne uniquement dans les versions 64 bits de 4D.

Description

La commande **Lire informations cache** retourne un objet contenant des informations détaillées sur le contenu actuel du cache (mémoire utilisée, tables et index chargés, etc.)

Note : Cette commande fonctionne uniquement en mode local (4D Server et 4D) ; elle ne doit pas être utilisée avec 4D en mode accès distant.

Par défaut, l'information retournée se réfère seulement à la base courante en cours d'exécution. Le paramètre objet optionnel *dbFilter* vous permet de spécifier la portée de cette commande :

- passez l'attribut "dbFilter" avec la valeur "All" pour obtenir les informations sur le cache de toutes les bases lancées, y compris les composants.
- passez l'attribut "dbFilter" avec la valeur "" (chaîne vide) pour obtenir des informations uniquement sur la base courante (équivalent à l'omission du paramètre *dbFilter*).

La commande **Lire informations cache** retourne un objet unique qui contient toutes les informations pertinentes à propos du cache. L'objet retourné a la structure suivante :

```
{
    "maxMem": Maximum cache size (real),
    "usedMem": Current cache size (real),
    "objects": [...] Array of objects currently loaded in cache
}
```

Les éléments du tableau *objects* sont des objets racine (tables, index, Blobs, etc.) qui sont actuellement chargés dans le cache. Chaque élément contient les attributs spécifiques qui décrivent son statut courant. Pour plus d'informations sur l'interprétation avancée de ces données, veuillez contacter les services techniques de 4D.

Exemple

Vous souhaitez obtenir des informations sur la base de données courante :

```
C_OBJET($cache)
$cache:=Lire informations cache
```

Vous souhaitez obtenir des informations sur la base courante et tous les composants ouverts :

```
C_OBJET($dbFilter)
OB FIXER($dbFilter;"dbFilter";"All")
$cache:=Lire informations cache($dbFilter)
```

LIRE STATISTIQUES MEMOIRE

LIRE STATISTIQUES MEMOIRE (typeInfo ; tabNoms ; tabValeurs ; tabNombre)

Paramètre	Type		Description
typeInfo	Entier long	⇨	Sélecteur d'information à obtenir
tabNoms	Tableau texte	⇦	Libellés des informations
tabValeurs	Tableau réel	⇦	Valeurs des informations
tabNombre	Tableau réel	⇦	Nombre d'objets concernés (si disponible)

Description

La commande **LIRE STATISTIQUES MEMOIRE** permet de récupérer des informations relatives à l'utilisation du cache de données par 4D. Ces informations peuvent être utiles à l'analyse du fonctionnement de l'application.

Passez dans le paramètre *typeInfo* une valeur indiquant le type d'information que vous souhaitez obtenir :

- 1 = Informations générales sur la mémoire. Ces informations sont également disponibles via l'Explorateur d'exécution : taille de mémoire physique, virtuelle, libre, occupée, mémoire pile (*stack memory*) et mémoire pile disponible (*free stack memory*)...
- 2 = Résumé des statistiques d'occupation du cache de la base de données (4D 32 bits seulement).

Note de compatibilité : la valeur 2 est obsolète pour les versions 64 bits de 4D.
Pour ces versions, nous recommandons l'utilisation de la commande **Lire informations cache** pour récupérer les statistiques du cache.

A l'issue de l'exécution de la commande, les statistiques demandées sont fournies dans les tableaux *tabNoms*, *tabValeurs* et *tabNombre*. Pour plus d'informations sur l'interprétation avancée de ces données, veuillez contacter le service technique de 4D SAS.

Lire taille cache

Lire taille cache -> Résultat

Paramètre	Type		Description
Résultat	Réel		Taille, en octets, du cache de la base de données

Description

La commande **Lire taille cache** retourne, en octets, la taille courante du cache de la base de données.
Note : Cette commande fonctionne uniquement en mode local (4D Server et 4D) ; elle ne doit pas être utilisée avec 4D en mode accès distant.

Exemple

Voir l'exemple de la commande **FIXER TAILLE CACHE**.

Glisser-Déposer

Présentation du Glisser-Déposer

4D dispose de fonctions intégrées vous permettant de gérer le glisser-déposer ("drag and drop") parmi les objets de vos formulaires et vos applications. Vous pouvez glisser-déposer un objet sur un autre objet situé dans la même fenêtre ou dans une autre fenêtre. Autrement dit, vous pouvez effectuer des glisser-déposer à l'intérieur d'un même process ou entre différents process.

Vous pouvez également glisser-déposer des objets entre les formulaires 4D et d'autres applications ou effectuer l'opération inverse. Par exemple, il est possible de glisser-déposer un fichier image GIF dans un champ image 4D. Il est également possible de sélectionner du texte dans une application de traitement de texte et de le déposer dans une variable texte de 4D.

Enfin, il est possible de déposer des objets directement sur l'application sans qu'un formulaire ne soit nécessairement au premier plan. La **Méthode base Sur déposer** permet dans ce cas de gérer le glisser-déposer. Ce principe permet par exemple d'ouvrir un document 4D Write en le déposant sur l'icône de l'application 4D.

Note : Pour ne pas alourdir cette introduction, nous admettrons ici que le glisser-déposer permet de "transporter" des données d'un point à un autre. Nous verrons plus loin qu'un glisser-déposer peut aussi être la métaphore (c'est-à-dire la représentation au niveau de l'interface utilisateur) de toute opération, quelle qu'elle soit.

Propriétés d'objets "Glissable" et "Déposable"

Si vous souhaitez qu'un objet soit **glissable**, c'est-à-dire que vous puissiez le faire glisser et le déposer sur un autre objet, vous devez sélectionner la propriété "Glissable"pour cet objet dans la Liste des propriétés. L'objet que vous faites glisser est appelé **objet source** de l'opération de glisser-déposer.

Si vous souhaitez qu'un objet soit **déposable**, c'est-à-dire que l'objet puisse être la destination d'une opération de glisser-déposer, vous devez sélectionner la propriété "Déposable" pour cet objet dans la Liste des propriétés. L'objet qui reçoit les données est appelé **objet de destination** de l'opération de glisser-déposer.

Glisser automatique et **Déposer automatique** : ces propriétés supplémentaires sont disponibles pour les champs et variables texte, combo box et list box. L'option **Déposer automatique** est également disponible pour les champs et variables image. Elles permettent d'activer un mode de glisser-déposer automatique basé sur la copie du contenu (le glisser-déposer n'est plus géré par les événements formulaires 4D). Reportez-vous au pararaphe "Glisser-Déposer automatique" à la fin de ce chapitre.

Par défaut, les objets nouvellement créés ne possèdent aucune de ces propriétés. Il est de votre ressort de les sélectionner explicitement.

Tous les objets situés dans un formulaire entrée ou dans une boîte de dialogue peuvent être définis comme glissables et déposables. Les éléments individuels d'un tableau (par exemple une zone de défilement), les éléments d'une liste hiérarchique ou les lignes d'une list box peuvent être glissé(e)s et déposé(e)s. Inversement, vous pouvez faire glisser et déposer tout objet sur un élément individuel d'un tableau ou d'une liste hiérarchique, ou encore une ligne de list box. Il n'est toutefois pas possible de faire glisser et de déposer des objets depuis la zone de corps d'un formulaire sortie.

Vous pouvez également gérer les glisser-déposer sur l'application, en-dehors de tout formulaire, via la **Méthode base Sur déposer**.

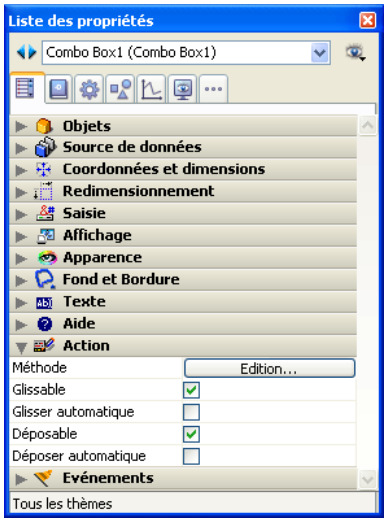
Afin de vous laisser "carte blanche" lors de la construction d'une interface utilisateur exploitant le glisser-déposer, 4D vous permet d'utiliser tout type d'objet actif (champ ou variable) en tant qu'objet source ou destination. Par exemple, si vous le souhaitez, vous pouvez glisser-déposer des boutons.

Notes

- Pour faire glisser un texte ou un bouton ayant la propriété "glissable", vous devez au préalable appuyer sur la touche **Alt** (Windows) ou **Option** (Mac OS).
- Par défaut, dans le cas des variables et champs image, l'image et sa référence sont glissées. Pour ne faire glisser que la référence de la variable ou du champ, appuyez au préalable sur la touche **Alt** (Windows) ou **Option** (Mac OS).
- Lorsque, pour un objet de type List box, les propriétés “Glissable” et “Ligne déplaçable” sont définies simultanément, la propriété “Ligne déplaçable” est prioritaire en cas de déplacement d’une ligne. Le glisser n’est pas possible dans ce cas.

Notez qu'un objet "glissable" et "déposable" peut être déposé sur lui-même (à moins que vous n'interdisiez cette opération, reportez-vous aux paragraphes suivants).

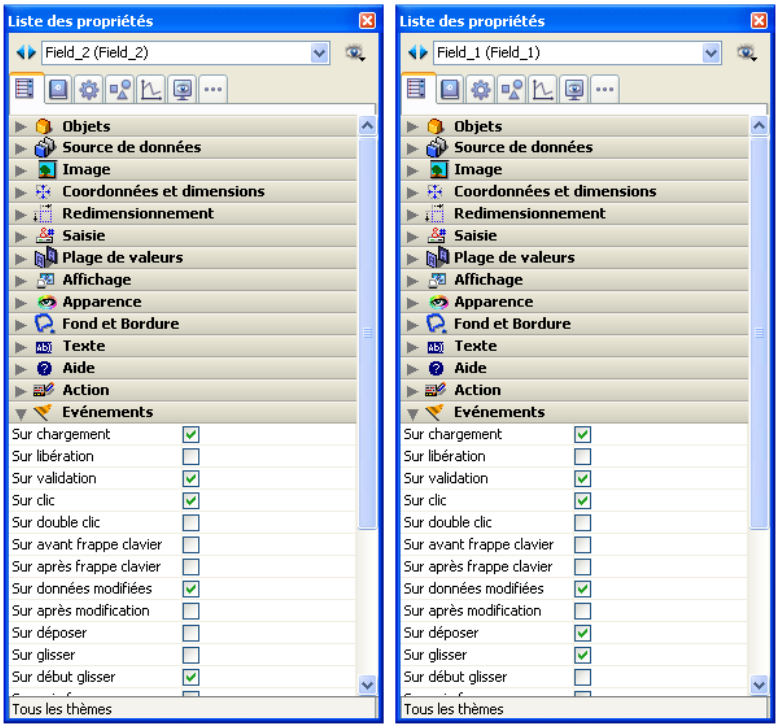
Voici la Liste des propriétés, dans laquelle les propriétés "Glissable" et "Déposable" ont été définies pour l'objet sélectionné :



Gérer le glisser-déposer par programmation

La gestion du glisser-déposer par programmation est basée sur trois événements formulaires : Sur début glisser, Sur glisser et Sur déposer.
A noter que l'événement Sur début glisser **est généré dans le contexte de l'objet source** du glisser tandis que Sur glisser et Sur déposer **sont envoyés à l'objet de destination uniquement**.

Pour que l'application puisse traiter ces événements, il doivent avoir été sélectionnés de manière appropriée dans la Liste des propriétés :



Sur début glisser

L'événement formulaire Sur début glisser est sélectionnable pour tous les objets de formulaire pouvant être glissés. Il est généré dans tous les cas lorsque l'objet dispose de la propriété **Glissable**.
A la différence de l'événement formulaire Sur glisser, Sur début glisser est appelé dans le contexte de l'objet à la source du glisser. Il peut être appelé dans la méthode de l'objet source ou la méthode du formulaire de l'objet source.
Cet événement est utile pour la gestion avancée du glisser. Il permet notamment de :

- lire les données et les signatures présentes dans le conteneur (via la commande **LIRE DONNEES CONTENEUR**).
- ajouter des données et des signatures dans le conteneur (via la commande **AJOUTER DONNEES AU CONTENEUR**).
- utiliser une icône personnalisée pendant l'action de glisser (via la commande **FIXER ICONE GLISSER**).
- accepter ou refuser le glisser via $0 dans la méthode de l'objet glissé. Pour signaler que le glisser est accepté, la méthode de l'objet source doit retourner *0* (zéro), vous exécutez donc *$0:=0*. Pour signaler que le glisser est refusé, la méthode de l'objet source doit retourner *-1* (moins un), vous exécutez donc *$0:=-1*. Si aucun résultat n'est retourné, 4D considère que le glisser est accepté.

Les données de 4D sont placées dans le conteneur de données avant l'appel de l'événement. Par exemple, pour un glisser sans l'option **Glisser automatique**, le texte glissé est déjà dans le conteneur au moment de l'appel de l'événement.

Sur glisser

L'événement Sur glisser est envoyé de manière répétée à l'objet de destination lorsque le pointeur de la souris est placé sur l'objet. Généralement, en réponse à cet événement, vous effectuez les actions suivantes :

- Vous appelez la commande **PROPRIETES GLISSER DEPOSER** qui vous renseigne sur l'objet source.
- En fonction de la nature et du type de l'objet de destination (celui duquel la méthode est en cours d'exécution) et de l'objet source, vous **acceptez** ou **refusez** le glisser.

Pour signaler que le glisser est accepté, la méthode de l'objet de destination doit retourner *0* (zéro), vous exécutez donc *$0:=0*. Pour signaler que le glisser est refusé, la méthode de l'objet de destination doit retourner *-1* (moins un), vous exécutez donc *$0:=-1*. Pendant un événement Sur glisser, 4D traite la méthode de l'objet comme une fonction. Si aucun résultat n'est retourné, 4D considère que le glisser est accepté.

Si vous acceptez le glisser, l'objet de destination est activé. Si vous le refusez, l'objet de destination reste inactivé. Accepter le glisser ne signifie pas que les données glissées vont être insérées dans l'objet de destination. Cela signifie uniquement que l'objet de destination, si le bouton de la souris était relâché à cet instant, accepterait les données.

Si vous ne gérez pas l'événement Sur glisser pour un objet dont la propriété "Déposable" a été sélectionnée, l'objet sera activé pour tous les glisser, quels que soient la nature et le type des données glissées.

La traitement de l'événement Sur glisser vous permet de contrôler la première phase d'une opération de glisser-déposer : non seulement vous pouvez tester si le type des données glissées est compatible avec l'objet de destination — et donc accepter ou refuser le glisser — mais également, vous en informez l'utilisateur, car 4D active ou non l'objet de destination en fonction de votre décision.

Le code traitant un événement Sur glisser doit être court et s'exécuter rapidement car cet événement est envoyé de manière répétée à l'objet de destination courant, en fonction des mouvements de la souris.

ATTENTION : A compter de la version 11 de 4D, si le glisser-déposer est un **glisser-déposer interprocess**, ce qui signifie que l'objet source est situé dans un process (fenêtre) différent de celui de l'objet de destination, la méthode de l'objet de destination lors de l'événement Sur glisser est exécutée **dans le contexte du process de destination**. Pour connaître la valeur des éléments glissés, vous devez utiliser les commandes de communication interprocess. Il est généralement préférable dans ce cas d'utiliser l'événement Sur début glisser et les commandes du thème **Conteneur de données**.

Sur déposer

L'événement Sur déposer est envoyé (une seule fois) à l'objet de destination lorsque le bouton de la souris est relâché alors que le pointeur se trouvait au-dessus de l'objet. Cet événement est la seconde phase d'un glisser-déposer, dans laquelle vous effectuez les véritables opérations répondant à l'action de l'utilisateur.

Cet événement n'est pas envoyé à l'objet si le glisser n'a pas été accepté dans le ou les événement(s) Sur glisser. Si vous traitez l'événement Sur glisser pour un objet et refusez le glisser, l'événement Sur déposer ne se déclenche pas. Ainsi, si pendant l'événement Sur glisser vous avez testé la compatibilité entre le type des données de l'objet source et de destination et accepté le glisser, vous n'avez pas besoin de tester de nouveau les données dans l'événement Sur déposer : vous savez déjà qu'elles sont compatibles.

L'aspect le plus intéressant de l'implémentation du glisser-déposer dans 4D est que le programme vous permet de faire ce que vous voulez. Par exemple :

- Si un élément de liste hiérarchique est déposé sur un champ de type Texte, vous pouvez décider d'insérer le texte de l'élément de liste au début, au milieu ou à la fin du champ de texte.
- Votre formulaire contient un bouton image à deux états représentant une corbeille vide et une corbeille pleine. Le glisser-déposer d'un objet sur ce bouton pourrait provoquer, du point de vue de l'interface utilisateur : “supprimer l'objet qui a été glissé-déposé dans la corbeille”. Ici, le glisser-déposer ne transporte pas des données d'un point à un autre, mais plutôt il déclenche une action.
- Glisser un élément de tableau depuis une palette flottante vers un objet dans un formulaire pourrait signifier “afficher dans cette fenêtre l'enregistrement du client dont le nom a été glissé-déposé depuis la fenêtre flottante listant les noms de tous les clients stockés dans la base”.
- Etc.

La gestion du glisser-déposer de 4D est une boîte à outils vous permettant d'implémenter toutes les métaphores d'interface utilisateur auxquelles vous pouvez penser.

Les commandes du thème Glisser-Déposer

La commande **PROPRIETES GLISSER DEPOSER** retourne :

- un pointeur vers l'objet glissé (champ ou variable),
- le numéro de l'élément de tableau ou de liste hiérarchique le cas échéant,
- le numéro du process source.

La commande **Position deposer** retourne le numéro ou la position de l'élément cible si l'objet de destination est un tableau (c'est-à-dire une zone de défilement), une liste hiérarchique, un texte ou une combo box, ainsi que le numéro de colonne si l'objet est une list box.

Les commandes telles que **RESOUDRE POINTEUR** et **Type** sont utiles pour tester la nature et le type de l'objet source.

Lorsque l'opération de glisser-déposer est destinée à copier les données glissées :

- Si le glisser-déposer est effectué à l'intérieur du même process, utilisez ces commandes pour effectuer les actions correspondantes (c'est-à-dire simplement assigner l'objet source à l'objet de destination).
- Si le glisser-déposer est interprocess, soyez vigilant lorsque vous accédez aux données glissées : vous devez récupérer l'instance des données située dans le process source. Si les données glissées proviennent d'une variable, utilisez la commande **LIRE VARIABLE PROCESS** pour obtenir la valeur correcte. Si les données glissées proviennent d'un champ, il est probable que l'enregistrement courant de la table n'est pas le même d'un process à l'autre, vous devez donc accéder au bon enregistrement. Il est généralement préférable dans ce cas d'utiliser les commandes du thème [#title id="81"/ et l'événement Sur début glisser.

Lorsque le glisser-déposer n'est pas destiné à déplacer des données mais plutôt à créer des métaphores d'interface pour des opérations particulières, vous pouvez virtuellement réaliser tout ce que vous voulez.

Les commandes du thème Conteneur de données

Si le glisser-déposer implique le déplacement de données hétérogènes ou de documents entre deux applications 4D ou une application 4D et une application tierce, les commandes du thème “Conteneur de données” vous fourniront tous les outils nécessaires.
En effet, ces commandes permettent de gérer à la fois le copier-coller et le glisser-déposer de données. 4D exploite deux conteneurs de données : l'un pour les données copiées (ou coupées), qui est en fait celui du Presse-papiers et l'autre pour les données en cours de glisser-déposer. Ces deux conteneurs sont gérés à l'aide des mêmes commandes. Vous accédez à l'un ou à l'autre en fonction du contexte.

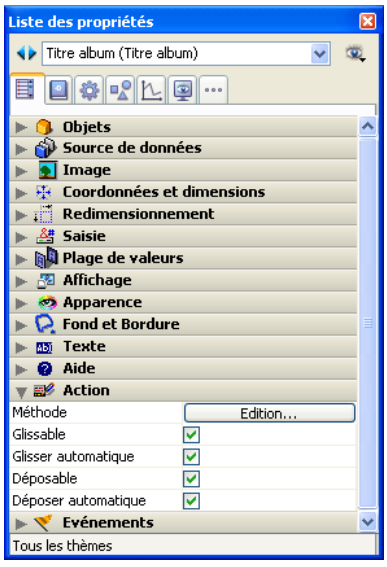
Pour plus d'informations sur l'utilisation des commandes du thème “Conteneur de données” dans le cadre du glisser-déposer, reportez-vous à la section **Gestion du conteneur de données**.

Glisser-Déposer automatique

Les objets texte (champs, variables, combo box et list box) ainsi que les objets image autorisent le glisser-déposer automatique, c'est-à-dire le déplacement ou la copie direct(e) de la sélection de texte ou d'une image d'une zone à une autre par simple clic. Il peut être employé dans la même zone 4D, entre deux zones 4D, ou entre 4D et une autre application, par exemple WordPad.

Note : En cas de glisser-déposer automatique entre deux zones 4D, les données sont déplacées, c'est-à-dire qu'elles sont supprimées de la zone source. Si vous souhaitez recopier les données, appuyez sur la touche **Ctrl** (Windows) ou **Option** (OS X) pendant l'opération (sous OS X, vous devez appuyer sur la touche **Option** *après* avoir commencé le glisser).

Le glisser-déposer automatique peut être paramétré séparément pour chaque objet d'un formulaire via deux options de la Liste des propriétés : **Glisser automatique** et **Déposer automatique** :



- **Glisser automatique** (objets de type texte uniquement) : Lorsque cette option est cochée, le mode de glisser automatique est activé pour l'objet. Dans ce mode, l'événement formulaire Sur début glisser n'est PAS généré. Si vous souhaitez “forcer” l’utilisation du glisser standard lorsque le glisser automatique est activé, appuyez sur la touche **Alt** (Windows) ou **Option** (OS X) pendant l’opération (sous OS X, vous devez appuyer sur la touche **Option** *avant* de commencer le glisser).
  Cette option n'est pas disponible pour les images.

- **Déposer automatique** : Cette option permet d’activer le mode de déposer automatique. Dans ce mode, 4D gère automatiquement — si possible — l’insertion des données glissées de type texte ou image et déposées sur l’objet (les données sont collées dans l’objet). Les événements Sur glisser et Sur déposer dans ce cas ne sont pas générés. En revanche, les événements Sur après modification (lors du déposer) et Sur données modifiées (lorsque l'objet perd le focus) sont générés.
  En cas de déposer de données autres que du texte ou des images (autre objet 4D, fichier, etc.) ou de données complexes, l’application se réfère à la valeur de l’option **Déposable** : si elle est cochée, les événements Sur glisser et Sur déposer sont générés, dans le cas contraire le déposer est refusé. Ce principe dépend également de la valeur de l’option “Interdire de glisser des données ne provenant pas de 4D” (cf. ci-dessous).

Interdire de glisser des données ne provenant pas de 4D (compatibilité)

A compter de la version 11, 4D permet le glisser-déposer de sélections, d’objets ou de fichiers extérieurs à 4D, comme par exemple des fichiers image. Cette possibilité doit être prise en charge par le code de la base.
Dans les bases de données converties depuis une version précédente de 4D, cette possibilité peut entraîner des dysfonctionnements si le code existant n’est pas adapté. Pour cette raison, une option de Préférences permet d'inactiver cette fonction : **Interdire de glisser des données ne provenant pas de 4D**. Cette option est placée dans la page Application/Compatibilité. Elle est cochée par défaut dans les bases converties.

Lorsque cette option est cochée, le déposer d’objets externes est refusé dans les formulaires 4D. A noter toutefois que l’insertion d’objets externes reste possible dans les objets disposant de l’option **Déposer automatique**, lorsque l’application peut interpréter les données déposées (texte ou image).

Méthode base Sur déposer

La **Méthode base Sur déposer** est disponible dans les applications 4D locales ou distantes.

Cette méthode base est exécutée automatiquement en cas de déposer d'objets dans l'application 4D en-dehors de tout formulaire ou fenêtre, c'est-à-dire :

- dans une zone vide de la fenêtre MDI (Windows),
- sur l'icône 4D dans le Dock (Mac OS) ou sur le Bureau du système.

Sous Mac OS, il est nécessaire de maintenir les touches **Option**+**Commande** enfoncées pendant le déposer afin que la méthode base soit appelée.

Lors d'un déposer sur l'icône de l'application 4D sur le bureau, la **Méthode base Sur déposer** est appelée uniquement si l'application est déjà lancée, sauf dans le cas des applications fusionnées avec 4D Desktop. Dans ce cas, la méthode base est appelée même lorsqu'elle n'est pas lancée. Ce principe permet de définir des signatures de documents personnalisées.

Exemple

Cet exemple permet d'ouvrir un document 4D Write déposé en-dehors de tout formulaire :

```
  `Méthode base Sur Déposer
fichierDéposé:=Lire fichier dans conteneur(1)
Si(Position(".4W7";fichierDéposé)=Longueur(fichierDéposé)-3)
   zexterne:=Creer fenetre externe(100;100;500;500;0;fichierDéposé;"_4D Write")
   WR OUVRIR DOCUMENT(zexterne;fichierDéposé)
Fin de si
```

FIXER ICONE GLISSER

FIXER ICONE GLISSER (icône {; décalageH {; décalageV}})

Paramètre	Type		Description
icône	Image	⇒	Icône à utiliser lors du glisser
décalageH	Entier long	⇒	Décalage horizontal du bord gauche de l'image par rapport à la position du curseur (>0 = vers la gauche, <0 = vers la droite)
décalageV	Entier long	⇒	Décalage vertical du bord supérieur de l'image par rapport à la position du curseur (>0 = vers le haut, <0 = vers le bas)

Description

La commande **FIXER ICONE GLISSER** vous permet d'associer l'image *icône* au curseur lors des glisser-déposer gérés par programmation.

Cette commande peut être appelée uniquement dans le contexte de l'événement formulaire Sur début glisser (cf. commande **Evenement formulaire**).

Passez dans le paramètre *icône* l'image à utiliser. Sa taille maximale est de 256x256 pixels. Si l'une de ses dimensions excède 256 pixels, elle est automatiquement redimensionnée.

Vous pouvez passer dans *décalageH* et *décalageV* des valeurs de décalage en pixels :

- passez dans *décalageH* le décalage horizontal du bord gauche de l'icône par rapport à la position du curseur. Passez une valeur positive pour appliquer le décalage vers la gauche ou une valeur négative pour appliquer le décalage vers la droite.
- passez dans *décalageV* le décalage vertical du bord supérieur de l'icône par rapport à la position du curseur. Passez une valeur positive pour appliquer le décalage vers le haut ou une valeur négative pour appliquer le décalage vers le bas.

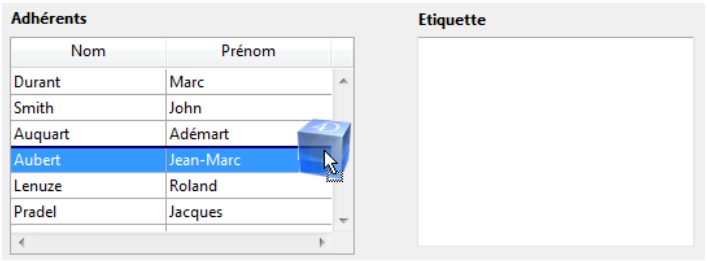
Si vous omettez ce paramètre, le curseur est placé au centre de l'icône.

Exemple

Dans un formulaire, l'utilisateur peut générer une étiquette par glisser-déposer d'une ligne. Dans la méthode objet de la list box, vous écrivez :

```
Si(Evenement formulaire=Sur début glisser)
      LIRE FICHIER IMAGE(Dossier 4D(Dossier Resources courant)+"splash.png";vpict)
      CREER IMAGETTE(vpict;vpict;48;48)
      FIXER ICONE GLISSER(vpict)
Fin de si
```

Lors du glisser d'une ligne, l'image apparaît :



A noter que vous pouvez modifier la position du curseur par rapport à l'image :

```
FIXER ICONE GLISSER(vpict;0;0)
```

Position deposer

Position deposer {(numColonne | posYImage)} -> Résultat

Paramètre	Type		Description
numColonne | posYImage	Entier long	⇐	Numéro de colonne de list box (-1 si le déposer a lieu après la dernière colonne) ou Position coordonnée Y dans l'image
Résultat	Entier long	⇒	• Numéro (tableau/list box) ou • Position (liste hiérarchique) ou • Position dans la chaîne (texte/combo box) de l'élément de destination ou • -1 si le déposer a lieu après le dernier élément de tableau ou de liste ou • Position coordonnée X dans l'image

Description

Position deposer permet de connaître l'emplacement, dans un objet de destination “complexe”, auquel un objet a été (glissé et) déposé. Généralement, vous utiliserez **Position deposer** pendant le traitement d'un événement glisser-déposer qui s'est produit dans un tableau, une list box, une liste hiérarchique, un champ texte ou une image.

- Si l'objet de destination est un tableau, la fonction retourne un numéro d'élément.
- Si l'objet de destination est une list box, la fonction retourne un numéro de ligne. Dans ce cas, la fonction retourne également dans le paramètre facultatif *numColonne* le numéro de la colonne sur laquelle le déposer a eu lieu.
- Si l'objet de destination est une liste hiérarchique, la fonction retourne une position d'élément.
- Si l'objet de destination est une variable ou un champ de type texte ou encore une combo box, la fonction retourne une position de caractère à l'intérieur de la chaîne.
  Dans tous les cas ci-dessus, la fonction retourne -1 si l'objet source a été déposé après le dernier élément de l'objet de destination.
- Si l'objet de destination est une variable ou un champ de type image, la fonction retourne l'emplacement horizontal du clic et, dans le paramètre facultatif *posYImage*, l'emplacement vertical du clic. Les valeurs retournées sont exprimées en pixels et relativement au système de coordonnées locales.

Si vous appelez **Position deposer** pendant le traitement d'un événement qui n'est pas de type glisser-déposer dans un tableau, une list box, une combo box, une liste hiérarchique, un texte ou une image, la fonction retourne également -1.

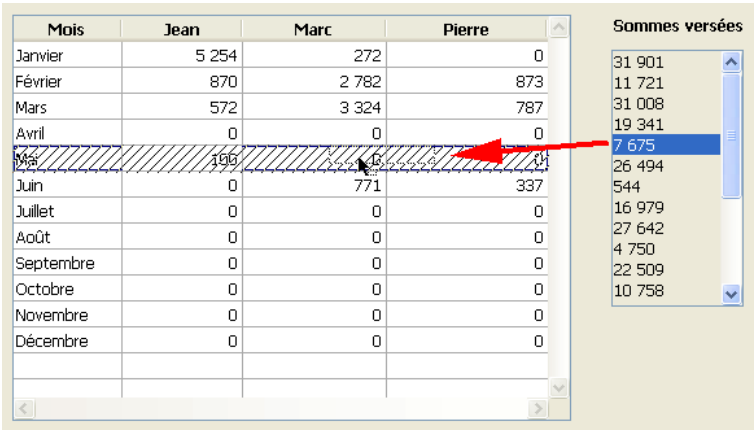
Rappel : Pour qu'un objet de formulaire accepte des données déposées, la propriété **Déposable** doit lui avoir été assignée. De plus, sa méthode objet doit être appelée par l'événement Sur glisser et/ou Sur déposer si vous voulez pouvoir gérer ce type d'événement.

Exemple 1

Reportez-vous aux exemples de la commande **PROPRIETES GLISSER DEPOSER**.

Exemple 2

Dans l’exemple suivant, une liste de sommes doit être ventilée par mois et par personne. L’opération s’effectue par glisser-déposer depuis une zone de défilement :



La méthode objet de la list box contient le code suivant :

```
Au cas ou
   :(Evenement formulaire=Sur glisser)
      PROPRIETES GLISSER DEPOSER($source;$lignetab;$numprocess)
      Si($source=Pointeur vers("ZD1")) `Si le déposer provient bien de la zone de défilement
         $0:=0
      Sinon
         $0:=-1 `On refuse le déposer
      Fin de si
   :(Evenement formulaire=Sur déposer)
      PROPRIETES GLISSER DEPOSER($source;$lignetab;$numprocess)
      $numligne:=Position deposer($numcol)
      Si($numcol=1)
         BEEP
      Sinon
         Au cas ou `Addition des valeurs déposées
            :($numcol=2)
```

```
                Jean{$numligne}:=Jean{$numligne}+ZD1{$lignetab}
            :($numcol=3)
                Marc{$numligne}:=Marc{$numligne}+ZD1{$lignetab}
            :($numcol=4)
                Pierre{$numligne}:=Pierre{$numligne}+ZD1{$lignetab}
         Fin de cas
         SUPPRIMER DANS TABLEAU(ZD1;$lignetab) `Mise à jour de la zone
      Fin de si
Fin de cas
```

PROPRIETES GLISSER DEPOSER

PROPRIETES GLISSER DEPOSER (srcObjet ; srcElément ; srcProcess)

Paramètre	Type		Description
srcObjet	Pointeur	⇐	Pointeur vers l'objet source du glisser-déposer
srcElément	Entier long	⇐	Numéro de l'élément de tableau glissé ou Numéro de la ligne de list box glissée ou Elément de la liste hiérarchique glissé ou -1 si l'objet glissé n'est ni un élément de tableau, ni une ligne de list box ni un élement de liste
srcProcess	Entier long	⇐	Numéro du process source

Description

La commande **PROPRIETES GLISSER DEPOSER** vous permet de récupérer des informations sur l'objet source lorsque l'événement Sur glisser ou Sur déposer est déclenché pour un objet “complexe” (tableau, list box ou liste hiérarchique).

Généralement, la commande **PROPRIETES GLISSER DEPOSER** se place dans la méthode objet (ou une des sous-méthodes qu'elle appelle) de l'objet pour lequel l'événement Sur glisser ou Sur déposer se produit.

Rappel : Des données peuvent être déposées sur un objet de formulaire si la propriété **Déposable** lui a été assignée. De plus, la méthode qui lui est associée doit être appelée par l'événement Sur déposer et/ou Sur glisser si vous voulez traiter ce type d'événement.

Après l'appel de cette commande :

- Le paramètre *srcObjet* est un pointeur vers l'objet source, c'est-à-dire l'objet qui a été glissé et déposé. Notez que cet objet peut être ou non identique à l'objet de destination, autrement dit l'objet pour lequel l'événement Sur déposer ou Sur glisser a été déclenché. Le glisser-déposer de valeurs entre des objets de même type est utile pour les tableaux et les listes hiérarchiques : cela vous fournit un moyen simple de permettre à l'utilisateur de trier manuellement un tableau ou une énumération.
- Si les données glissées-déposées sont un élément de tableau (l'objet source étant un tableau), le paramètre *srcElément* est égal au numéro de cet élément. Si les données glissées-déposées sont une ligne de list box, le paramètre *srcElément* est égal au numéro de cette ligne. Si les données glissées-déposées sont un élément de liste hiérarchique, le paramètre *srcElément* retourne la position de cet élément. Sinon, si l'objet source n'appartient à aucune de ces catégories, *srcElément* est égal à -1.
- Des opérations de glisser-déposer peuvent être effectuées entre différents process. Le paramètre *srcProcess* est égal au numéro du process auquel appartient l'objet source. Il est important de tester la valeur de ce paramètre. En effet, vous pouvez traiter un glisser-déposer “process” en copiant simplement les données source dans l'objet de destination. En revanche, lorsque vous traitez un glisser-déposer “interprocess”, vous devez utiliser la commande **LIRE VARIABLE PROCESS** pour récupérer les données source à partir de l'instance de l'objet du process source. Notez que généralement, le glisser-déposer dans une interface utilisateur s'effectue à partir de variables (tableaux et liste hiérarchiques) vers des zones de saisie de données (champs ou variables).

Note de compatibilité : Depuis la version 11 de 4D, il est recommandé de gérer les opérations de glisser-déposer, notamment interprocess, à l'aide de l'événement Sur début glisser et des commandes du thème **Conteneur de données**.

Si vous appelez **PROPRIETES GLISSER DEPOSER** alors qu'aucun événement glisser-déposer ne s'est produit, *srcObjet* retourne un pointeur NIL, *srcElément* retourne -1 et *srcProcess* retourne 0.

Astuce : 4D gère pour vous l'aspect graphique du glisser-déposer. Mais c'est à vous de traiter l'événement de manière appropriée. Dans les exemples ci-dessous, le traitement consiste à copier les données qui ont été glissées. Mais vous pouvez également implémenter des interfaces plus sophistiquées dans lesquelles, par exemple, le glisser-déposer d'un élément de tableau depuis une palette flottante provoque le remplissage de la fenêtre de destination (la fenêtre dans laquelle se trouve l'objet de destination) avec des données structurées (comme plusieurs champs provenant d'un enregistrement désigné par l'élément de tableau source).

Vous pouvez appeler la commande **PROPRIETES GLISSER DEPOSER** lors de l'événement formulaire Sur glisser afin de décider si l'objet de destination doit ou non accepter l'opération, en fonction du type et/ou de la nature de l'objet source (ou pour toute autre raison). Si vous acceptez le glisser-déposer, la méthode de l'objet doit retourner *$0:=0*. Si vous n'acceptez pas l'opération, la méthode de l'objet doit retourner *$0:=-1*. L'acceptation ou le refus d'un glisser-déposer est visible à l'écran : l'objet sera ou ne sera pas sélectionnable (par exemple encadré) en tant que destination du glisser-déposer.

Exemple 1

Vous disposez, dans plusieurs formulaires de votre base, de zones de défilement. Vous voulez que l'utilisateur puisse réordonner manuellement les éléments des zones par simple glisser-déposer à l'intérieur de chaque zone. Plutôt que d'écrire du code spécifique pour chaque cas, vous voulez utiliser une méthode projet générique qui traite toutes les zones de défilement. Pour cela, vous pouvez écrire :

```
   ` Méthode projet de traitement de glisserdéposer interne dans un tableau
   ` Traitement de glisserdéposer interne dans un tableau ( Pointeur ) -> Booléen
   ` Traitement de glisserdéposer interne dans un tableau ( -> Tableau ) -> Est un glisser-déposer interne dans un tableau

 Au cas ou
    :(Evenement formulaire=Sur glisser)
       PROPRIETES GLISSER DEPOSER($vpSrcObj;$vlSrcElem;$vlPID)
       Si($vpSrcObj=$1)
  `Accepter le glisser-déposer s'il est interne au tableau
          $0:=0
       Sinon
          $0:=-1
       Fin de si
    :(Evenement formulaire=Sur déposer)
   ` Récupérer les informations sur l'objet source du glisser-déposer
       PROPRIETES GLISSER DEPOSER($vpSrcObj;$vlSrcElem;$vlPID)
   ` Récupérer le numéro de l'élément de destination
       $vlDstElem :=Position deposer
   ` Si l'élément n'a pas été glissé-déposé sur lui-même
       Si($vlDstElem #$vlSrcElem)
```

```
   ` Stocker l'élément glissé dans l'élément 0 du tableau
         $1->{0}:=$1->{$vlSrcElem}
   ` Effacer l'élément glissé
         SUPPRIMER DANS TABLEAU($1->;$vlSrcElem)
   ` Si l'élément de destination est au-delà de l'élément glissé
         Si($vlDstElem>$vlSrcElem)
   ` Décrémenter le numéro de l'élément de destination
            $vlDstElem:=$vlDstElem-1
         Fin de si
   ` Si le glisser-déposer s'est produit au-delà du dernier élément
         Si($vlDstElem=-1)
   ` Définir le numéro de l'élément de destination comme un nouvel élément ajouté à la fin du tableau
            $vlDstElem:=Taille tableau($1->)+1
         Fin de si
   ` Insérer ce nouvel élément
         INSERER DANS TABLEAU($1->;$vlDstElem)
   ` Fixer sa valeur, préalablement stockée dans l'élément zéro du tableau
         $1->{$vlDstElem}:=$1->{0}
   ` L'élément devient le nouvel élément sélectionné du tableau
         $1->:=$vlDstElem
      Fin de si
Fin de cas
```

Une fois que vous avez écrit cette méthode projet, vous pouvez l'utiliser ainsi :

```
   ` Méthode objet Zone de défilement unTableau

Au cas ou
   `...
   :(Evenement formulaire=Sur glisser)
      $0:=Traitement de glisserdéposer interne dans un tableau(Self)
   :(Evenement formulaire=Sur déposer)
      Traitement de glisserdéposer interne dans un tableau(Self))
   `...
Fin de cas
```

Exemple 2

Vous disposez, dans plusieurs formulaires de votre base, de zones de texte saisissables. Vous voulez que l'utilisateur puisse y saisir des données par glisser-déposer à partir de sources multiples. Plutôt que d'écrire du code spécifique pour chaque cas, vous voulez utiliser une méthode projet générique qui traite toutes les zones de texte. Pour cela, écrivez la méthode suivante :

```
   ` Méthode projet Traitement du déposer dans variable Texte
   ` Traitement du déposer dans variable Texte ( Pointeur )
   ` Traitement du déposer dans variable Texte ( -> variable texte ou chaîne )

Au cas ou
   ` Utilisation de cet événement pour accepter ou refuser le glisser-déposer
   :(Evenement formulaire=Sur glisser)
   ` Initialiser $0 pour le refus
      $0:=-1
   ` Récupérer les informations sur l'objet source du glisser-déposer
      PROPRIETES GLISSER DEPOSER($vpSrcObj;$vlSrcElem;$vlPID)
   ` Dans cet exemple, nous refusons le glisser-déposer d'un objet sur lui-même
      Si($vpSrcObj#$1)
   ` Récupérer le type des données glissées
         $vlSrcType:=Type($vpSrcObj->)
         Au cas ou
            :($vlSrcType=Est un texte)
   ` OK pour les variables texte
               $0:=0
            :($vlSrcType=Est une variable chaîne)
   ` OK pour les variables chaîne
               $0:=0
            :(($vlSrcType=Est un tableau chaîne)|($vlSrcType=Est un tableau texte))
   ` OK pour les tableaux chaîne et texte
               $0:=0
            :(($vlSrcType=Est un entier long)|($vlSrcType=Est un numérique))
               Si(Liste existante($vpSrcObj->))
   ` OK pour les liste hiérarchiques
                  $0:=0
               Fin de si
         Fin de cas
      Fin de si

   ` Utilisation de cet événement pour effectuer réellement l'action de glisser-déposer
   :(Evenement formulaire=Sur déposer)
      $vtDonnéesGlissées:=""
   ` Récupérer les informations sur l'objet source du glisser-déposer
```

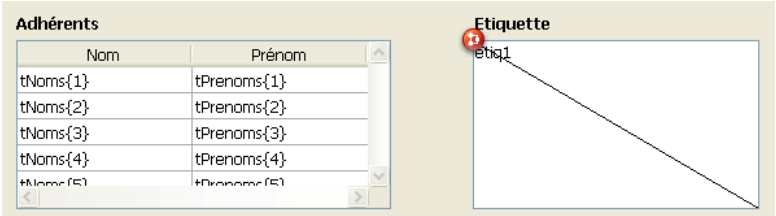
```
      PROPRIETES GLISSER DEPOSER($vpSrcObj;$vlSrcElem;$vlPID)
   ` Récupérer le type des données glissées
      $vlSrcType:=Type($vpSrcObj->)
      Au cas ou
   ` Si c'est un tableau
         :(($vlSrcType=Tableau chaîne)|($vlSrcType=TABLEAU TEXTE))
            Si($vlPID#Numero du process courant)
   ` Lire l'élément depuis l'instance de la variable dans le process source
               LIRE VARIABLE PROCESS($vlPID;$vpSrcObj->{$vlSrcElem};$vtDraggedData)
            Sinon
   ` Glisser-déposer depuis le même process, copions juste la valeur
               $vtDraggedData:=$vpSrcObj->{$vlSrcElem}
            Fin de si
   ` Si c'est une liste
         :(($vlSrcType=Est un numérique) | ($vlSrcType=Est un entier long))
   ` Si c'est une liste en provenance d'un autre process
            Si($vlPID#Numero du process courant)
   `Récupérer la référence de la liste dans l'autre process
               LIRE VARIABLE PROCESS($vlPID;$vpSrcObj->;$vlList)
            Sinon
               $vlList:=$vpSrcObj->
            Fin de si
   ` Si la liste existe
            Si(Liste existante($vpSrcObj->))
   `Récupérer le texte de l'élément dont on a obtenu la position
               INFORMATION ELEMENT($vlList;$vlSrcElem;$vlItemRef;$vsItemText)
               $vtDraggedData:=$vsItemText
            Fin de si
         Sinon
   ` C'est une variable chaîne ou texte
            Si($vlPID#Numero du process courant)
               LIRE VARIABLE PROCESS($vlPID;$vpSrcObj->;$vtDraggedData)
            Sinon
               $vtDraggedData:=$vpSrcObj->
            Fin de si
      Fin de cas
   ` S'il y a effectivement quelque chose à déposer (l'objet source pourrait être vide)
      Si($vtDraggedData # "")
         $1->:=$1->+$vtDraggedData
      Fin de si
Fin de cas
```

Une fois que vous avez écrit cette méthode projet, vous pouvez l'utiliser ainsi :

```
   ` Méthode objet du champ de texte [uneTable]unTexte

Au cas ou
         ` ...
   :(Evenement formulaire=Sur glisser)
      $0:=Traitement du déposer dans variable texte(Self)

   :(Evenement formulaire=Sur déposer)
      Traitement du déposer dans variable texte(Self)
         ` ...

Fin de cas
```

Exemple 3

Nous souhaitons remplir une zone de texte (par exemple une étiquette) avec des données glissées depuis une list box.

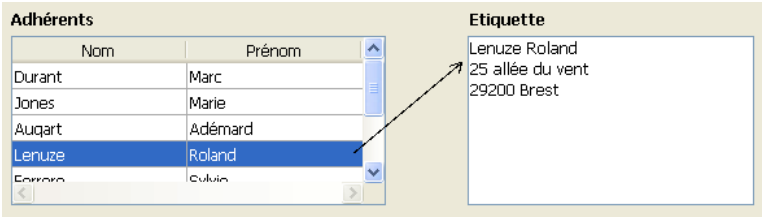


Voici la méthode de l'objet etiq1 :

```
Au cas ou
   :(Evenement formulaire=Sur glisser)
      PROPRIETES GLISSER DEPOSER($source;$lignetab;$numprocess)
      Si($source=Pointeur vers("list box1"))
         $0:=0 `On accepte le glisser
      Sinon
         $0:=-1 `On refuse le glisser
```

```
      Fin de si
   :(Evenement formulaire=Sur déposer)
      PROPRIETES GLISSER DEPOSER($source;$lignetab;$numprocess)
      CHERCHER([Adhérents];[Adhérents]Nom=tNoms{$lignetab})
      Si(Enregistrements trouves([Adhérents])#0)
         etiq1:=[Adhérents]Nom+" "+[Adhérents]Prénom+Caractere(Retour chariot)+[Adhérents]Adresse+Caractere(Retour
chariot)+[Adhérents]Code postal+" "+[Adhérents]Ville
      Fin de si
 Fin de cas
```

Il est dès lors possible d'effectuer l'action suivante :



Graphes

GRAPHE
PARAMETRES DU GRAPHE
_o_GRAPHE SUR SELECTION

GRAPHE

GRAPHE (graphImage ; graphNum | graphParams ; xCatégories {; zValeurs} {; zValeurs2 ; ... ; zValeursN})

Paramètre	Type		Description
graphImage	Variable image	➡	Variable image
graphNum | graphParams	Entier long, Objet	➡	Entier long : Numéro de type de graphe, Objet (64 bits uniquement) : Paramètres du graphe
xCatégories	Tableau	➡	Catégories sur l'axe des x
zValeurs	Tableau	➡	Valeurs à représenter graphiquement (jusqu'à 8 valeurs)

Description

***Note de compatibilité** : A compter de 4D v14, la commande* GRAPHE *fonctionne uniquement avec une variable image en premier paramètre. La syntaxe obsolète utilisant une zone de graphe (4D Chart) n'est plus prise en charge.*

La commande **GRAPHE** crée un graphe dans une variable image à partir de valeurs provenant de tableaux.

Les graphes générés par cette commande sont dessinés via le moteur de rendu SVG intégré. Ils bénéficient des fonctions d'interface associées aux variables images : menu contextuel en mode Application (permettant notamment le choix du format d'affichage), barres de défilement, etc.

Note : SVG (Scalable Vector Graphics) est un format de fichier graphique vectoriel (extension .svg). Basé sur le XML, ce format est largement répandu et peut être notamment affiché par les navigateurs Web. Pour plus d'informations, reportez-vous à http://www.w3.org.Graphics/SVG/. La commande **SVG EXPORTER VERS IMAGE** vous permet également de tirer parti du moteur SVG intégré.

Passez dans le paramètre *graphImage* le nom de la variable image devant afficher le graphe dans le formulaire.

Le second paramètre définit le type de graphe à utiliser. Vous disposez de deux possibilités :

- passer un paramètre *graphNum* de type *Entier long* (toutes versions de 4D) : dans ce cas, vous devez passer un nombre entre 1 et 8. Les différents types de graphes disponibles sont listés dans l'exemple présenté plus bas. Une fois le graphe créé, vous pouvez modifier son type en modifiant la valeur de *graphNum* et en exécutant de nouveau la commande **GRAPHE**. Vous pouvez par la suite modifier certaines caractéristiques du graphe en appelant la commande **PARAMETRES DU GRAPHE**. Voir Exemple 1.
- passer un paramètre *graphParams* de type *Objet* (versions 4D 64 bits uniquement, hormis 4D Server Windows 64 bits) : dans ce cas, vous devez passer un objet qui contient les diverses propriétés du graphe que vous souhaitez définir. Pour cela, vous pouvez utiliser les constantes placées dans le thème "**Paramètre des graphes**" (cf ci-dessous). Cette syntaxe vous permet de définir le type de graphe ainsi que tous ses paramètres spécifiques (légende, xmin, etc.) en un seul appel. Avec ce principe, vous pouvez sauvegarder les graphes générés en tant qu'images SVG et les afficher dans un navigateur standard tel que FireFox, Chrome, IE ou Safari (les graphes générés sont conformes au SVG standard implémenté dans les navigateurs). En outre, cette syntaxe donne accès à de nombreux paramètres, vous permettant de personnaliser, entre autres, l'espacement entre les barres, les marges, les couleurs de barres, etc. Voir Exemples 2, 3 et 4. Attention, si vous utilisez cette syntaxe, la commande **PARAMETRES DU GRAPHE** ne doit PAS être appelée.

Le paramètre *xCatégories* définit les valeurs qui seront utilisées pour l'axe des X. Vous pouvez passer un champ de type chaîne, Heure, Date, ou un type numérique. Il doit y avoir le même nombre d'éléments de tableau dans *xCatégories* qu'il y en a dans chaque *zValeurs*.

Le paramètre *zValeurs* définit les valeurs à représenter graphiquement. Elles doivent être de type numérique. Vous pouvez passer jusqu'à huit ensembles de données. Les graphes en secteurs ne représentent que le premier *zValeurs*.

IDs automatiques

Des IDs spécifiques sont automatiquement attribués aux éléments présents dans le graphe SVG :

IDs	Description
ID_graph_1 à ID_graph_8	Colonnes, lignes, aires...
ID_graph_shadow_1 à ID_graph_shadow_8	Ombre des colonnes, lignes, aires...
ID_bullet_1 à ID_bullet_8	Points (*graphes en Lignes et en Points uniquement*)
ID_pie_label_1 à ID_pie_label_8	Libellés des secteurs (*graphes en Secteurs uniquement*)
ID_legend	Légende
ID_legend_1 à ID_legend_8	Titres des légendes
ID_legend_border	Encadrement des légendes
ID_legend_border_shadow	Ombre des encadrements des légendes
ID_x_values	Valeurs axe des X
ID_y_values	Valeurs axe des Y
ID_y0_axis	Valeurs axe des Z
ID_background	Arrière plan
ID_background_shadow	Ombre de l'arrière plan
ID_x_grid	Grille sur l'axe des X
ID_x_grid_shadow	Ombre de la grille sur l'axe des X
ID_y_grid	Grille sur l'axe des Y
ID_y_grid_shadow	Ombre de la grille sur l'axe des Y

Attributs graphParams

Lorsque vous utilisez le paramètre *graphParams*, vous devez passer un objet qui contient les diverses propriétés du graphe que vous souhaitez définir. Pour cela, vous pouvez utiliser les constantes suivantes, placées dans le thème "**Paramètre des graphes**" :

Constante	Type	Valeur	Comment
Graphe afficher la légende	Chaîne	displayLegend	**Valeurs possibles** : Booléen **Valeur par défaut** : Vrai
Graphe couleur fond	Chaîne	graphBackgroundColor	**Valeurs possibles :** Expression couleur norme SVG (texte), par exemple "#7F8E00", "Pink" ou "#0a1414" **Valeurs possibles** : Expression couleur SVG (texte), par exemple "#7F8E00", "Pink" ou

Constante	Type	Valeur	Comment
Graphe couleur fond document	Chaîne	documentBackgroundColor	"#0a1414". Lorsqu'un graphe enregistré sous forme d'image SVG est ouvert dans une autre application, la couleur de fond du document est affichée uniquement si le moteur de rendu SVG prend en charge la norme *SVG tiny 1.2* (prise en charge sur IE, Firefox mais pas sur Chrome).
Graphe couleur ombre	Chaîne	graphBackgroundShadowColor	**Valeurs possibles** : Expression couleur SVG (texte), par exemple "#7F8E00", "Pink" ou "#0a1414"
Graphe couleur police	Chaîne	fontColor	**Valeurs possibles** : Expression couleur SVG (texte), par exemple "#7F8E00", "Pink" ou "#0a1414".
Graphe couleur police légende	Chaîne	legendFontColor	**Valeurs possibles** : Expression couleur SVG (texte), par exemple "#7F8E00", "Pink" ou "#0a1414"
Graphe couleurs	Chaîne	colors	**Valeurs possibles** : Tableau texte. Couleurs pour chaque série de graphe. **Valeurs par défaut** : Blue-green (#19BAC9), Yellow (#FFC338), Purple (#573E82), Green (#4FA839), Orange (#D95700), Blue (#1D9DF2), Yellow-green (#B5CF32), Red (#D43A26)
Graphe décalage secteurs	Chaîne	pieShift	**Valeurs possibles** : Réels **Valeur par défaut** : 8
Graphe épaisseur des lignes	Chaîne	lineWidth	**Valeurs possibles** : Réels **Valeur par défaut** : 2
Graphe espacement colonnes	Chaîne	columnGap	**Valeurs possibles** : Entier long **Valeur par défaut** : 12 Définit l'espacement entre les colonnes du graphe
Graphe espacement icônes légende	Chaîne	legendIconGap	**Valeurs possibles** : Réel **Valeur par défaut** : Graphe hauteur icônes légende/2
Graphe hauteur des points	Chaîne	plotHeight	**Valeurs possibles** : Réels **Valeur par défaut** : 12
Graphe hauteur icônes légende	Chaîne	legendIconHeight	**Valeurs possibles** : Réels **Valeur par défaut** : 20
Graphe hauteur par défaut	Chaîne	defaultHeight	**Valeurs possibles** : Réel **Valeur par défaut** : 400. Si graphType=7 (secteurs), valeur défaut = 600
Graphe largeur des points	Chaîne	plotWidth	**Valeurs possibles** : Réel **Valeur par défaut** : 12
Graphe largeur icônes légende	Chaîne	legendIconWidth	**Valeurs possibles** : Réel **Valeur par défaut** : 20
Graphe largeur max colonne	Chaîne	columnWidthMax	**Valeurs possibles** : Réel **Valeur par défaut** : 200
Graphe largeur min colonne	Chaîne	columnWidthMin	**Valeurs possibles** : Réel **Valeur par défaut** : 10
Graphe largeur par défaut	Chaîne	defaultWidth	**Valeurs possibles** : Réel **Valeur par défaut** : 600. Si graphType=7 (Secteurs), valeur par défaut = 800
Graphe libellés légende	Chaîne	legendLabels	**Valeurs possibles** : Tableau texte. S'il est manquant, 4D affiche des icônes sans texte.
Graphe marge basse	Chaîne	bottomMargin	**Valeurs possibles** : Nombre réel **Valeur par défaut** : 12
Graphe marge droite	Chaîne	rightMargin	**Valeurs possibles** : Réels **Valeur par défaut** : 2
Graphe marge gauche	Chaîne	leftMargin	**Valeurs possibles** : Réel **Valeur par défaut** : 12
Graphe marge haute	Chaîne	topMargin	**Valeurs possibles** : Réels **Valeur par défaut** : 2
Graphe opacité fond	Chaîne	graphBackgroundOpacity	**Valeurs possibles** : Entier long entre 0 et 100 **Valeur par défaut** : 100
Graphe opacité fond document	Chaîne	documentBackgroundOpacity	**Valeurs possibles** : Entier long (0 à 100). Lorsqu'un graphe enregistré sous forme d'image SVG est ouvert dans une autre application, l'opacité du fond du document est affichée uniquement si le moteur de rendu SVG prend en charge la norme *SVG tiny 1.2* (prise en charge sur IE, Firefox mais pas sur Chrome). **Valeur par défaut :** 100
Graphe			

Constante	Type	Valeur	Commentaire
Cons dans points	Chaîne	dotRadius	**Valeurs possibles** : Réels **Valeur par défaut** : 12
Graphe taille police	Chaîne	fontSize	**Valeurs possibles** : Entier long **Valeur par défaut** : 12. Si graphType=7 (Secteurs), voir Graphe taille police des secteurs
Graphe taille police des secteurs	Chaîne	pieFontSize	**Valeurs possibles** : Réels **Valeur par défaut** : 16
Graphe type	Chaîne	graphType	**Valeurs possibles** : Entier long [1 à 8] où 1 = colonnes, 2 = colonnes proportionnelles, 3 = colonnes empilées, 4 = lignes, 5 = aires, 6 = points, 7 = secteurs, 8 = images. **Valeur par défaut** : 1 Si la valeur est nulle, le graphe n'est pas dessiné et aucun message d'erreur n'est affiché. Si la valeur est trop grande, le graphe n'est pas dessiné et un message d'erreur est affiché. Si vous souhaitez modifier les graphes de type image (valeur=8), vous devez recopier le dossier 4D/Resources/GraphTemplates/Graph_8_Pictures/ dans le dossier Resources de votre base et effectuer les modifications nécessaires. Les fichiers image locaux seront utilisés au lieu des fichiers de 4D. Les noms des fichiers image sont libres ; 4D trie les fichiers contenus dans le dossier et affecte le premier fichier au premier graphe. Ces fichiers peuvent être de type SVG ou image.
Graphe xGrille	Chaîne	xGrid	**Valeurs possibles** : Booléen **Valeur par défaut** : Vrai Utilisé uniquement avec les types proportionnels 4 et 6
Graphe xMax	Chaîne	xMax	**Valeurs possibles** : Nombre, Date, Heure (même type que le paramètre *xCatégories*). Seules les valeurs inférieures à xMax sont affichées dans le graphe. xMax est utilisé uniquement pour les graphes de type 4, 5, or 6 si xProp=Vrai et si *xCatégories* est de type numérique, date ou heure. Si ce paramètre est manquant ou si xMin>xMax, 4D calcule automatiquement la valeur xMax.
Graphe xMin	Chaîne	xMin	**Valeurs possibles** : Nombre, Date, Heure (même type que le paramètre *xCatégories*). Seules les valeurs supérieures xMin sont affichées dans le graphe. xMin est utilisé uniquement pour les graphes de type 4, 5, or 6 si xProp=Vrai et si *xCatégories* est de type numérique, date ou heure. Si ce paramètre est manquant ou si xMin>xMax, 4D calcule automatiquement la valeur xMin.
Graphe xProp	Chaîne	xProp	**Valeurs possibles** : Booléen **Valeur par défaut** : Faux Vrai pour un axe x proportionnel, Faux pour un axe x normal. Utilisé uniquement avec les types proportionnels 4, 5 et 6
Graphe yGrille	Chaîne	yGrid	**Valeurs possibles** : Booléen **Valeur par défaut** : Vrai
Graphe yMax	Chaîne	yMax	**Valeurs possibles** : Nombres Si ce paramètre est manquant, 4D calcule automatiquement la valeur yMax.
Graphe yMin	Chaîne	yMin	**Valeurs possibles** : Nombres Si ce paramètre est manquant, 4D calcule automatiquement la valeur yMin.

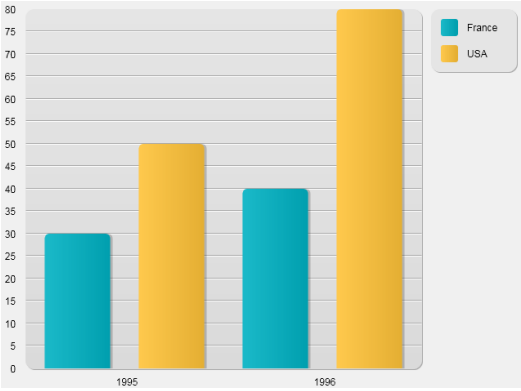
Exemple 1

Syntaxe avec *graphNum* : l'exemple suivant illustre les différents types de graphes que vous pouvez obtenir. Ce code doit être placé dans la méthode formulaire (ou une méthode objet) du formulaire contenant la variable image. A noter que, dans notre exemple, les données représentées sont constantes, ce qui n'est généralement pas le cas :

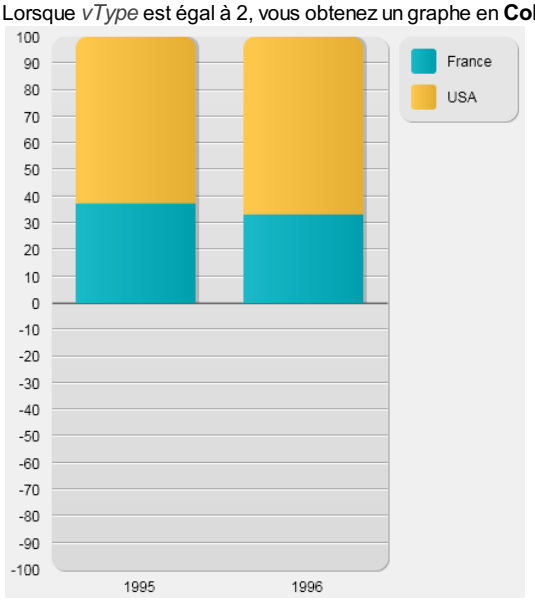
```
C_IMAGE(vGraph) //Variable du graphe
TABLEAU TEXTE(X;2) //Création d'un tableau pour l'axe des X
X{1}:="1995" //Libellé X #1
X{2}:="1996" //Libellé X #2
TABLEAU REEL(A;2) //Création d'un tableau pour l'axe des Z
A{1}:=30 // Insertion des données
A{2}:=40
TABLEAU REEL(B;2) //Création d'un second tableau pour l'axe des Z
B{1}:=50 // Insertion des données
B{2}:=80
vType:=1 //Initialisation du type de graphe
GRAPHE(vGraph;vType;X;A;B) //Dessiner le graphe
PARAMETRES DU GRAPHE(vGraph;0;0;0;0;Faux;Faux;Vrai;"France";"USA") //Définition des légendes du graphe
```

Les images suivantes représentent les graphes résultants :

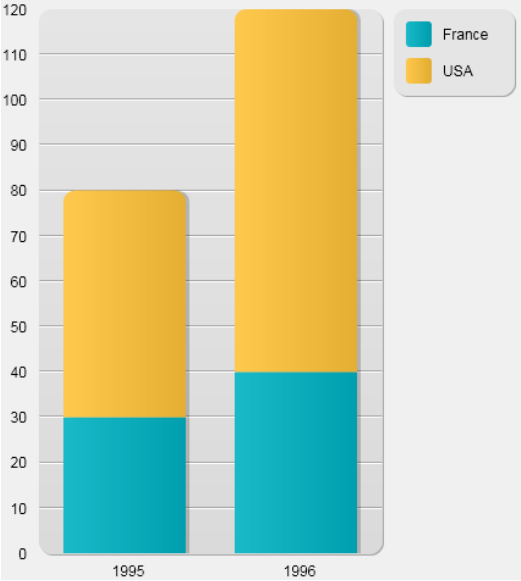
- Lorsque *vType* est égal à 1, vous obtenez un graphe en **Colonnes** :



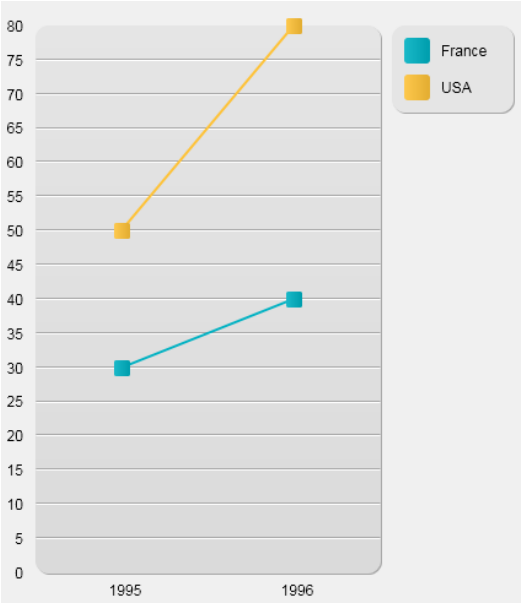
- Lorsque *vType* est égal à 2, vous obtenez un graphe en **Colonnes proportionnelles** :



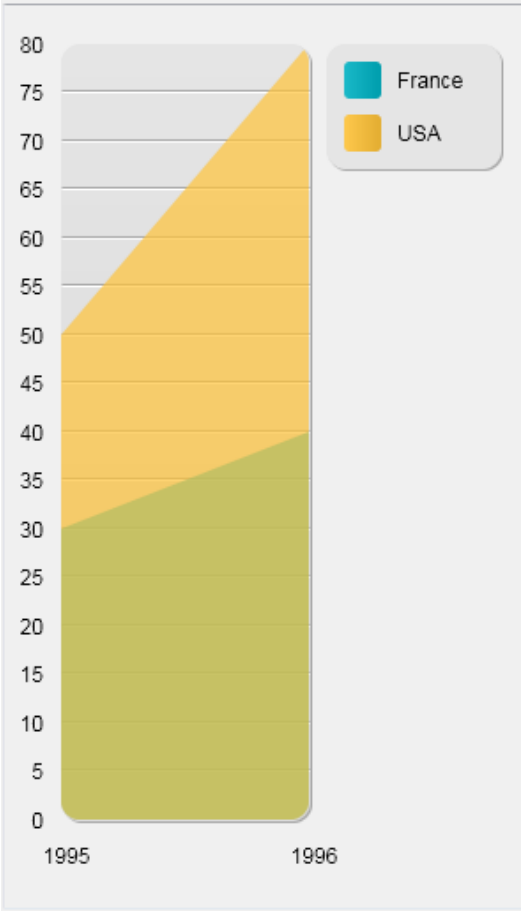
- Lorsque *vType* est égal à 3, vous obtenez un graphe en **Colonnes empilées :**



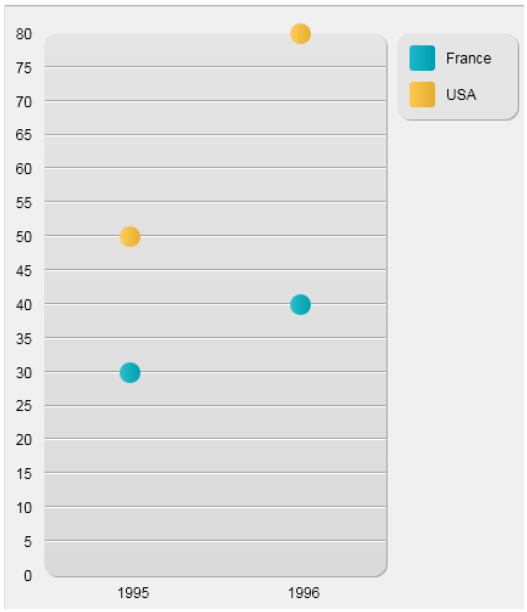
- Lorsque *vType* est égal à 4, vous obtenez un graphe en **Lignes :**



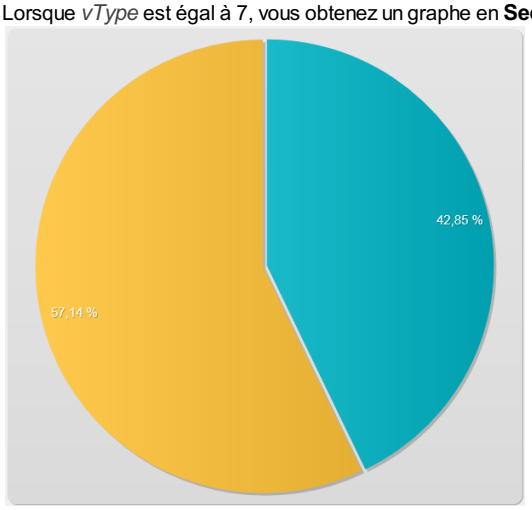
- Lorsque *vType* est égal à 5, vous obtenez un graphe en **Aires :**



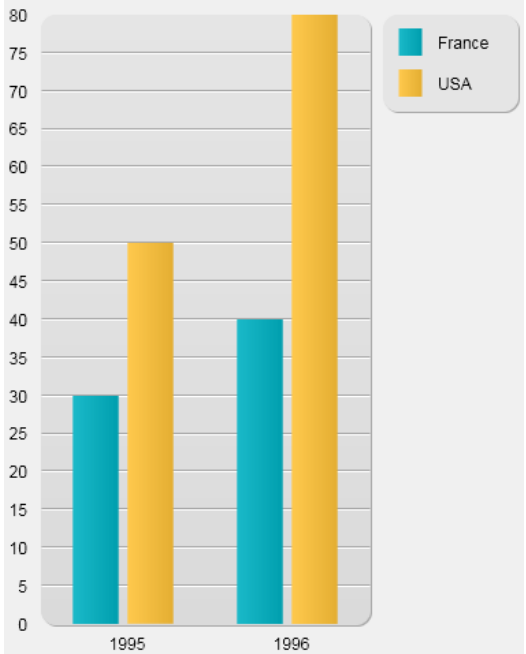
- Lorsque *vType* est égal à 6, vous obtenez un graphe en **Points :**



- Lorsque *vType* est égal à 7, vous obtenez un graphe en **Secteurs :**



- Lorsque *vType* est égal à 8, vous obtenez un graphe en **Images :**



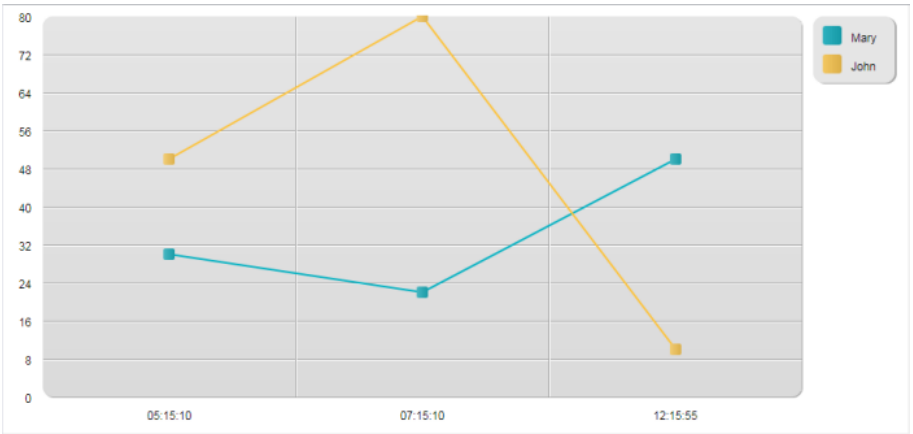
Note : Les images sont des rectangles simples par défaut.

Exemple 2

Syntaxe avec *graphParams* : Avec l'exemple suivant, vous dessinez un simple graphe en lignes basé sur des valeurs de temps :

```
C_IMAGE(vGraph) //Variable graphe
TABLEAU HEURE(X;3) //Tableau pour l'axe des X
X{1}:=?05:15:10? //libellé X n°1
```

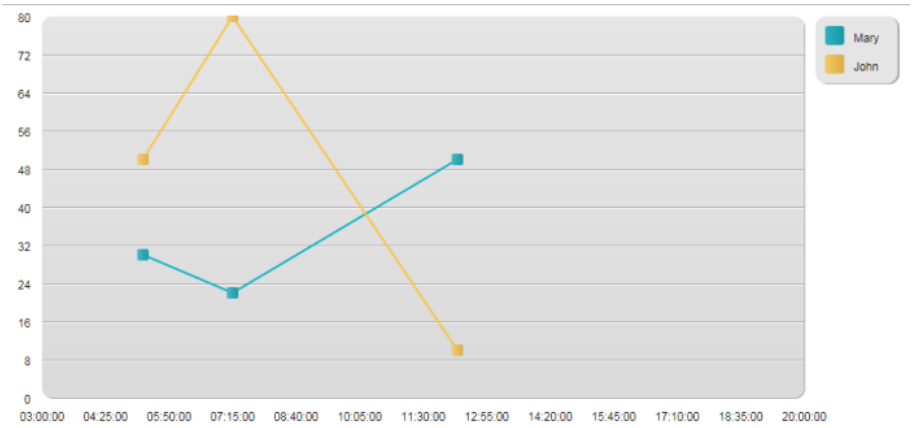
```
X{2}:=?07:15:10? //libellé X n°2
X{3}:=?12:15:55? //libellé X n°3

TABLEAU REEL(A;3) //Tableau pour l'axe des Y
A{1}:=30 //On ajoute quelques données
A{2}:=22
A{3}:=50

TABLEAU REEL(B;3) //Un autre ableau pour l'axe des Y
B{1}:=50 //On ajoute quelques données
B{2}:=80
B{3}:=10

C_OBJET(vSettings) //Initialisation des paramètres du graphe

OB FIXER(vSettings;Graphe type;4) //Type lignes

TABLEAU TEXTE(aLabels;2) //Définition des légendes pour le graphe
aLabels{1}:="Mary"
aLabels{2}:="John"
OB FIXER TABLEAU(vSettings;Graphe libellés légende;aLabels)

GRAPHE(vGraph;vSettings;X;A;B) //On dessine le graphe
```



Exemple 3

Avec les mêmes valeurs, on ajoute des paramètres personnalisés pour obtenir une vue différente :

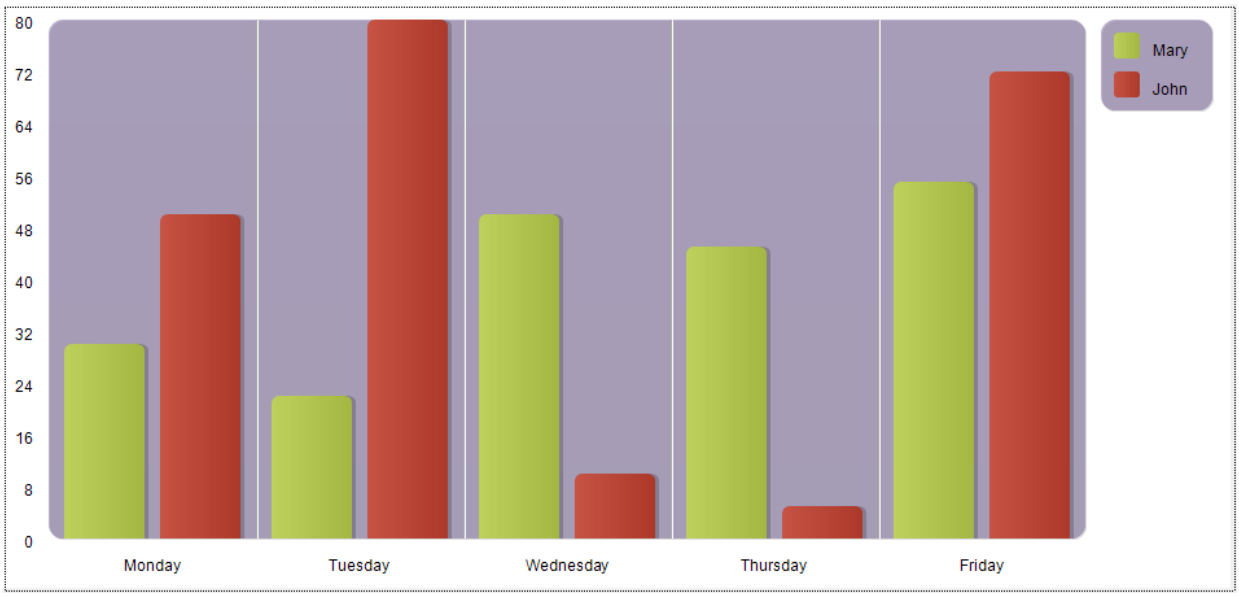
```
C_IMAGE(vGraph) //Variable graphe
TABLEAU HEURE(X;3) //Tableau pour l'axe des X
X{1}:=?05:15:10? //libellé X n°1
X{2}:=?07:15:10? //libellé X n°2
X{3}:=?12:15:55? //libellé X n°3

TABLEAU REEL(A;3) //Tableau pour l'axe des Y
A{1}:=30 //On ajoute quelques données
A{2}:=22
A{3}:=50

TABLEAU REEL(B;3) //Un autre ableau pour l'axe des Y
B{1}:=50 //On ajoute quelques données
B{2}:=80
B{3}:=10

C_OBJET(vSettings) //initialisation des paramètres du graphe

OB FIXER(vSettings;Graphe type;4) //Type lignes

TABLEAU TEXTE(aLabels;2) //On définit les légendes du graphe
aLabels{1}:="Mary"
aLabels{2}:="John"
OB FIXER TABLEAU(vSettings;Graphe libellés légende;aLabels)

 //options
OB FIXER(vSettings;Graphe xProp;Vrai) //proportionnel
OB FIXER(vSettings;Graphe xGrille;Faux) //on enlève la grille verticale
OB FIXER(vSettings;Graphe xMin;?03:00:00?) //on définit les limites
OB FIXER(vSettings;Graphe xMax;?20:00:00?)

GRAPHE(vGraph;vSettings;X;A;B) //On dessine le graphe
```



Exemple 4

Dans cet exemple, on personnalise divers paramètres :

```
C_IMAGE(vGraph) //variable du graphe
TABLEAU TEXTE(X;5) //Création d'un tableau pour l'axe des X
X{1}:="Monday" // 1er libellé X
X{2}:="Tuesday" // 2e libellé X
X{3}:="Wednesday" //etc.
X{4}:="Thursday"
X{5}:="Friday"

TABLEAU ENTIER LONG(A;5) //Création d'un tableau pour l'axe des Y
A{1}:=30 //On ajoute quelques données
A{2}:=22
A{3}:=50
A{4}:=45
A{5}:=55

TABLEAU ENTIER LONG(B;5) //Création d'un autre tableau pour l'axe des Y
B{1}:=50 //On ajoute quelques données
B{2}:=80
B{3}:=10
B{4}:=5
B{5}:=72

C_OBJET(vSettings) //Initialisation des paramètres du graphe

OB FIXER(vSettings;Graphe type;1) //graphe en barres

TABLEAU TEXTE(aLabels;2) //Définit les légendes pour le graphe
aLabels{1}:="Mary"
aLabels{2}:="John"
OB FIXER TABLEAU(vSettings;Graphe libellés légende;aLabels)

 //options
OB FIXER(vSettings;Graphe yGrille;Faux) //on enlève la grille verticale
OB FIXER(vSettings;Graphe couleur fond;"#573E82") //on définit une couleur de fond
OB FIXER(vSettings;Graphe opacité fond;40)
TABLEAU TEXTE($aTcols;2) //on définit les couleurs du graphe
$aTcols{1}:="#B5CF32"
$aTcols{2}:="#D43A26"
OB FIXER TABLEAU(vSettings;Graphe couleurs;$aTcols)
GRAPHE(vGraph;vSettings;X;A;B) //On dessine le graphe
```


80
72
64
56
48
40
32
24
16
8
0
Monday
Tuesday
Wednesday
Thursday
Friday
Mary
John

PARAMETRES DU GRAPHE

PARAMETRES DU GRAPHE (graphImage ; xmin ; xmax ; ymin ; ymax ; xprop ; grilleX ; grilleY ; titre {; titre2 ; ... ; titreN})

Paramètre	Type		Description
graphImage	Variable image	⇒	Variable image
xmin	Entier long, Date, Heure	⇒	Valeur minimale de l'échelle des X pour graphe proportionnel (lignes ou points)
xmax	Entier long, Date, Heure	⇒	Valeur maximale de l'échelle des X pour graphe proportionnel (lignes ou points)
ymin	Entier long	⇒	Valeur minimale de l'échelle des Y
ymax	Entier long	⇒	Valeur maximale de l'échelle des Y
xprop	Booléen	⇒	VRAI pour l'échelle des X proportionnelle ; FAUX pour l'échelle des X normale (lignes ou points)
grilleX	Booléen	⇒	VRAI pour la grille sur l'axe des X ; FAUX pour pas de grille sur l'axe des X (seulement si xprop est VRAI)
grilleY	Booléen	⇒	VRAI pour la grille sur l'axe des Y; FAUX pour pas de grille sur l'axe des Y
titre	Chaîne	⇒	Titre(s) pour les titre(s) des série(s)

Description

La commande **PARAMETRES DU GRAPHE** permet de paramétrer les échelles et les grilles d'un graphe placé dans un formulaire. Le graphe doit déjà avoir été défini à l'aide de la commande **GRAPHE**. **PARAMETRES DU GRAPHE** ne fait rien s'il s'agit d'un graphe de type secteurs. Cette commande doit impérativement être appelée dans le même process que le formulaire.

Note : Vous ne devez pas appeler cette commande si le graphe a été généré par la commande **GRAPHE** avec le paramètre *graphParams* de type *Objet*. Reportez-vous à la description de la commande **GRAPHE** pour plus d'informations.

Les paramètres *xmin*, *xmax*, *ymin* et *ymax* fixent les valeurs minimales et maximales pour les axes des X ou Y. Si la valeur des deux paramètres correspondants au même axe est nulle (0, ?00:00:00? ou !00/00/00! selon le type de données), les valeurs de graphe par défaut seront utilisées. Les paramètres *xmin* et *xmax* ne sont pris en compte que pour les graphes proportionnels (*xprop* est **Vrai**).
Le paramètre *xprop* fixe l'axe des X comme proportionnel (sont concernés par cette option les graphes de type 4 et 6). Lorsque ce paramètre est **Vrai**, chaque point sera placé sur l'axe des X par rapport aux valeurs des points si elles sont de type numérique, heure ou date.

Note : Avec 4D Server 64 bits OS X, le paramètre *xprop* est également pris en compte pour les graphes de type 5.

Les paramètres *grilleX* et *grilleY* montrent ou cachent les grilles. Une grille pour l'axe des X sera affichée s'il s'agit d'un graphe en points ou en lignes proportionnel.

Le(s) paramètre(s) *titre* spécifient les titres des légendes.

Exemple

Reportez-vous à l'exemple de la commande **GRAPHE**.

_o_GRAPHE SUR SELECTION

_o_GRAPHE SUR SELECTION
Ne requiert pas de paramètre

Description

***Commande supprimée** : Cette commande n'est plus prise en charge à compter de 4D v14.*

Images

- Introduction aux images
- BLOB VERS IMAGE
- COMBINER IMAGES
- CONVERTIR IMAGE
- CREER IMAGETTE
- ECRIRE FICHIER IMAGE
- ECRIRE IMAGE DANS BIBLIOTHEQUE
- Est un fichier image
- FIXER METADONNEES IMAGE
- FIXER NOM FICHIER IMAGE
- IMAGE VERS BLOB
- IMAGE VERS GIF
- Images egales
- LIRE FICHIER IMAGE
- LIRE FORMATS IMAGE Nouveauté 16.0
- LIRE IMAGE DANS BIBLIOTHEQUE
- LIRE METADONNEES IMAGE
- LIRE MOTS CLES IMAGE
- Lire nom fichier image
- LISTE CODECS IMAGES
- LISTE IMAGES DANS BIBLIOTHEQUE
- PROPRIETES IMAGE
- SUPPRIMER IMAGE DANS BIBLIOTHEQUE
- Taille image
- TRANSFORMER IMAGE
- *_o_ENREGISTRER IMAGE*
- *_o_LISTE TYPES IMAGES*
- *_o_QT CHARGER ET COMPRESSER IMAGE*
- *_o_QT COMPRESSER FICHIER IMAGE*
- *_o_QT COMPRESSER IMAGE*

Formats natifs pris en charge

4D intègre une gestion native des images. Cela signifie que les images sont affichées et stockées dans leur format d'origine, sans interprétation dans 4D. Les spécificités des différents formats (ombrages, zones transparentes...) sont conservées en cas de copier-coller et affichées sans altération. Cette prise en charge native est valide pour toutes les images stockées dans 4D : images de la bibliothèque, images collées dans les formulaires en mode Développement, images collées dans les champs ou variables en mode Application, etc.

4D utilise des API natives pour encoder et décoder les images (champs et variables) sous Windows et Mac OS. Ces implémentations donnent accès à de nombreux formats natifs, dont le format RAW, couramment utilisé par les appareils photo numériques.

- **Sous Windows**, 4D utilise WIC (Windows Imaging Component). WIC prend en charge nativement les formats BMP, PNG, ICO (décodage seulement), JPEG, GIF, TIFF et WDP (Microsoft Windows Digital Photo).
Il est possible d'utiliser des formats supplémentaires tels que JPEG-2000 en installant des codecs WIC tiers.
- **Sous Mac OS**, 4D utilise ImageIO. Tous les codecs ImageIO disponibles sont donc pris en charge nativement pour le décodage (lecture) ainsi que l'encodage (écriture) :

Décodage	Encodage
public.jpeg	public.jpeg
com.compuserve.gif	com.compuserve.gif
public.png	public.png
public.jpeg-2000	public.jpeg-2000
com.nikon.raw-image	public.tiff
com.pentax.raw-image	com.adobe.photoshop.image
com.sony.arw-raw-image	com.adobe.pdf
com.adobe.raw-image	com.microsoft.bmp
public.tiff com.canon.crw-raw-image	com.truevision.tga-image
com.canon.cr2-raw-image	com.sgi.sgi-image
com.canon.tif-raw-image	com.apple.pict *(obsolète)*
com.sony.raw.image	com.ilm.openexr-image
com.olympus.raw-image	
com.konicaminolta.raw-image	
com.panasonic.raw-image	
com.fuji.raw-image	
com.adobe.photoshop-image	
com.adobe.illustrator.ai-image	
com.adobe.pdf	
com.microsoft.ico	
com.microsoft.bmp	
com.truevision.tga-image	
com.sgi.sgi-image	
com.apple.quicktime-image *(obsolète)*	
com.apple.icns	
com.apple.pict *(obsolète)*	
com.apple.macpaint-image	
com.kodak.flashpix-image	
public.xbitmap-image	
com.ilm.openexr-image	
public.radiance	

Sous Windows comme sous Mac OS, les formats pris en charge varient en fonction du système d'exploitation et des codecs personnalisés installés sur les postes. Pour connaître les codecs disponibles, vous devez utiliser la commande **LISTE CODECS IMAGES**.

Note : WIC et ImageIO permettent l'utilisation de métadonnées dans les images. Deux commandes, **FIXER METADONNEES IMAGE** et **LIRE METADONNEES IMAGE** vous permettent d'en bénéficier dans vos développements.

Identifiants de codecs d'images

Les formats d'images reconnus par 4D sont retournés par la commande **LISTE CODECS IMAGES** sous forme d'identifiants de codecs d'images. Ces identifiants peuvent être :

- une extension (par exemple ".gif")
- un type Mime (par exemple "image/jpg")

La forme utilisée pour chaque format dépend du mode de déclaration du codec au niveau du système d'exploitation.
La plupart des commandes de gestion d'images de 4D attendent un identifiant de codec en paramètre. Il est donc impératif d'utiliser l'identifiant système retourné par la commande **LISTE CODECS IMAGES**.

Format d'image non disponible

Une icône spécifique est affichée pour les images stockées dans un format non disponible sur le poste. L'extension du format manquant est inscrite en bas de l'icône :

L'icône est automatiquement utilisée partout où l'image doit être affichée :

Cette icône indique que l'image ne peut être ni affichée ni manipulée localement -- mais elle peut être stockée sans altération pour être affichée sur une autre machine. C'est le cas, par exemple, pour les images PDF sous Windows ou les images basées sur PICT avec un 4D Server 64-bits sous OS X.

Activation de QuickTime (compatibilité)

Par défaut, les codecs d'image liés à QuickTime ne sont plus pris en charge dans 4D à compter de la v14.

Pour des raisons de compatibilité, vous pouvez toutefois réactiver QuickTime dans votre application à l'aide de l'option Prise en charge QuickTime de la commande **FIXER PARAMETRE BASE**. Il est toutefois déconseillé désormais d'utiliser les codecs QuickTime.

Note : L'option de réactivation de QuickTime est ignorée dans les versions 64 bits de 4D Developer Edition (pas de prise en charge de QuickTime).

Coordonnées de la souris dans une image

4D permet de récupérer les coordonnées locales de la souris dans un champ ou une variable image en cas de clic ou de survol, même si un défilement ou un zoom a été appliqué à l'image. Ce mécanisme, proche de celui d'une image map, peut être utilisé par exemple pour gérer des barres de boutons défilables ou l'interface de logiciels de cartographie.

Les coordonnées sont retournées dans les **Variables système MouseX** et **MouseY**. Les coordonnées sont exprimées en pixels par rapport à l'angle supérieur gauche de l'image (0,0). Lorsque la souris se trouve en-dehors du système de coordonnées de l'image, la valeur -1 est retournée dans **MouseX** et **MouseY**.

Vous pouvez lire la valeur des variables **MouseX** et **MouseY** dans le contexte des événements formulaire Sur clic, Sur double clic, Sur relâchement bouton, Sur début survol et Sur survol.

Opérateurs sur les images

4D vous permet d'effectuer des **opérations** sur les images 4D, telles que la concaténation, la superposition, etc. Ce point est traité dans la section **Opérateurs sur les images**.

BLOB VERS IMAGE

BLOB VERS IMAGE (blobImage ; image {; codec})

Paramètre	Type		Description
blobImage	BLOB	⇨	BLOB contenant une image
image	Image	⇦	Champ ou variable image 4D
codec	Chaîne	⇨	Identifiant de codec d'image

Description

La commande **BLOB VERS IMAGE** place dans un champ ou une variable image 4D une image stockée dans un BLOB, quel que soit son format initial.

Le fonctionnement de cette commande est analogue à celui de la commande **LIRE FICHIER IMAGE** ; elle s'applique simplement à un BLOB et non à un fichier. Elle permet d'afficher à tout moment des images stockées en format natif dans des BLOBs à l'aide, par exemple, de la commande **DOCUMENT VERS BLOB** ou **IMAGE VERS BLOB**.

Vous passez dans le paramètre *blobImage* le BLOB contenant l'image. L'image peut être de tout format pris en charge en natif par 4D. Vous pouvez obtenir la liste des formats disponibles à l'aide de la commande **LISTE CODECS IMAGES**. Si vous passez le paramètre facultatif *codec*, 4D utilisera la valeur fournie dans ce paramètre pour décoder le BLOB (voir ci-dessous le fonctionnement spécifique de la commande avec ce troisième paramètre).

Vous passez dans le paramètre *image* la variable ou le champ 4D de type image devant afficher l'image.

Note : Le format interne de l'image sera conservé au sein de la variable ou du champ 4D.

Après l'exécution de la commande, si le BLOB a pu être correctement décodé, l'*image* contient l'image affichable dans 4D.

Le paramètre facultatif *codec* vous permet de préciser le codec à utiliser pour décoder le BLOB.
Si vous passez dans codec un *codec* reconnu par 4D (retourné par la commande **LISTE CODECS IMAGES**), il est appliqué au BLOB et l'image est retournée dans le champ ou la variable *image*.
Si vous passez dans *codec* un codec non reconnu par 4D, un nouveau codec est enregistré dynamiquement avec l'identifiant passé en paramètre. 4D retourne alors une image qui encapsule le BLOB et la variable OK prend la valeur 1. Dans ce cas, pour récupérer le BLOB, il sera nécessaire d'utiliser la commande **IMAGE VERS BLOB** avec le même identifiant personnalisé. Ce mécanisme particulier permet de répondre à deux besoins spécifiques :

- encapsulation d'un BLOB (qui n'est pas une image) dans une image,
- chargement d'une image sans disposer du codec.
  La mise en oeuvre de ces mécanismes permet de notamment de créer des "tableaux de BLOBs" en passant par des tableaux images. Cette technique doit être utilisée avec précaution car, les tableaux étant entièrement chargés en mémoire, la manipulation de BLOBs de grande taille peut altérer le fonctionnement de l'application.

Note : Un BLOB créé par la commande **VARIABLE VERS BLOB** est géré automatiquement, il n'est pas nécessaire de passer le codec pour l'encapsuler, le BLOB étant "signé". Pour l'opération inverse dans ce cas, vous devez passer ".4DVarBlob" comme identifiant de codec à la commande **IMAGE VERS BLOB**.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1. En cas d'échec (absence de QuickTime, format d'image inconnu, paramètre *codec* reconnu mais ne validant pas le BLOB...), OK prend la valeur 0 et le champ ou la variable image 4D est retourné(e) vide.

COMBINER IMAGES

COMBINER IMAGES (imageRésultat ; image1 ; opérateur ; image2 {; décalHoriz ; décalVert})

Paramètre	Type		Description
imageRésultat	Image	⇦	Image résultant de la combinaison
image1	Image	⇨	Première image à combiner
opérateur	Entier long	⇨	Type de combinaison à effectuer
image2	Image	⇨	Seconde image à combiner
décalHoriz	Entier long	⇨	Décalage horizontal pour la superposition
décalVert	Entier long	⇨	Décalage vertical pour la superposition

Description

La commande **COMBINER IMAGES** permet de combiner les images *image1* et *image2* en mode *opérateur* pour en produire une troisième, *imageRésultat*. L'image résultat est de type composé et conserve toutes les caractéristiques des images sources.

Note : Cette commande reprend et étend les fonctionnalités proposées par les opérateurs "classiques" de transformation d'images (+/, etc., cf. section **Opérateurs sur les images**). Ces opérateurs restent parfaitement utilisables dans 4D.

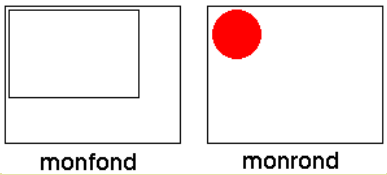
Passez dans *opérateur* le type de combinaison à appliquer. Trois types de combinaisons sont proposés, accessibles via des constantes placées dans le thème "**Transformation des images**" :

- Concaténation horizontale (1) : *image2* est accolée à *image1*, le coin supérieur gauche de *image2* coïncidant avec le coin supérieur droit de *image1*.
- Concaténation verticale (2) : image2 est accolée à *image1*, le coin supérieur gauche de *image2* coïncidant avec le coin inférieur gauche de *image1*.
- Superposition (3) : *image2* est placée par-dessus *image1*, le coin supérieur gauche de *image2* coïncidant avec le coin supérieur gauche de *image1*. Si les paramètres facultatifs *décalHoriz* et *décalVert* sont utilisés, une translation est appliquée à *image2* avant la superposition. Les valeurs passées dans *décalHoriz* et *décalVert* doivent correspondre à des pixels. Passez des valeurs positives pour un décalage vers la droite ou vers le bas et une valeur négative pour un décalage vers la gauche ou vers le haut.

Note : La superposition effectuée par la commande **COMBINER IMAGES** diffère de la superposition proposée par les opérateurs "classiques" & et | (superposition exclusive et superposition inclusive). Tandis que la commande **COMBINER IMAGES** conserve les caractéristiques de chaque image source dans l'image résultante, les opérateurs & et | traitent chaque pixel et génèrent une image bitmap dans tous les cas. Ces opérateurs, conçus à l'origine pour les images monochromes, sont désormais obsolètes.

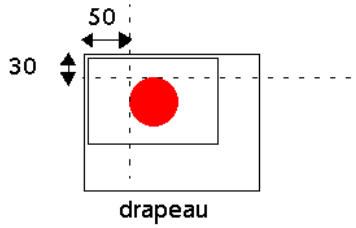
Exemple

Soient les images suivantes :




```
COMBINER IMAGES(drapeau;monfond;Superposition;monrond;50;30)
```

Résultat :



CONVERTIR IMAGE

CONVERTIR IMAGE (image ; codec {; compression})

Paramètre	Type		Description
image	Image	⇒	Image à convertir
		⇐	Image convertie
codec	Chaîne	⇒	Identifiant de codec d'image
compression	Réel	⇒	Qualité de compression

Description

La commande **CONVERTIR IMAGE** convertit *image* dans un nouveau type.

Le paramètre *codec* indique le type d'image à générer. Un codec peut être une extension (par exemple ".gif") ou un type Mime (par exemple "image/jpg"). Vous pouvez obtenir la liste des codecs disponibles via la commande **LISTE CODECS IMAGES**.

Si le champ ou la variable *image* est de type composé (si par exemple elle est issue d'un copier-coller), seules les informations correspondant au type *codec* sont conservées dans l'image résultante.

Note : Si le type de *codec* demandé est égal au type d'origine de l'*image*, aucune conversion n'est effectuée et l'image est retournée telle quelle (sauf si le paramètre *compression* est utilisé, cf. ci-dessous).

Le paramètre optionnel *compression*, s'il est passé, permet de définir la qualité de compression à appliquer à l'image résultante lorsqu'un codec compatible est utilisé. Passez dans *compression* une valeur entre 0 et 1 définissant la qualité de compression, 0 étant la qualité la plus médiocre (compression élevée) et 1 la meilleure qualité (compression faible). Ce paramètre est pris en compte uniquement lorsque le codec supporte la compression (par exemple JPEG ou HDPhoto) et est pris en charge par les APIs WIC et ImageIO. Pour plus d'informations sur les APIs de gestion d'image dans 4D, reportez-vous à la section **Introduction aux images**. Par défaut, si vous omettez le paramètre *compression*, la meilleure qualité est appliquée (compression = 1).

Exemple 1

Conversion de l'image vpPhoto au format jpeg :

```
CONVERTIR IMAGE(vpPhoto;".jpg")
```

Exemple 2

Conversion d'une image avec une qualité de 60 % :

```
CONVERTIR IMAGE(vPicture;".JPG";0,6)
```

CREER IMAGETTE

CREER IMAGETTE (source ; dest {; largeur {; hauteur {; mode {; profondeur}}}})

Paramètre	Type		Description
source	Image	⇒	Champ ou variable image 4D à passer en imagette
dest	Image	⇐	Imagette résultante
largeur	Entier	⇒	Largeur de l'imagette en pixels, Par défaut = 48
hauteur	Entier	⇒	Hauteur de l'imagette en pixels, Par défaut = 48
mode	Entier	⇒	Mode de création de l'imagette Par défaut = proportionnelle centrée (6)
profondeur	Entier	⇒	Obsolète, ne pas utiliser

Description

La commande **CREER IMAGETTE** retourne une imagette à partir d'une image source. Les imagettes sont généralement utilisées pour la prévisualisation d'images dans le cadre d'applications multimédia ou de sites Web.

Passez dans *source* la variable ou le champ image 4D contenant l'image source à réduire sous forme d'imagette, et dans *dest* la variable ou le champ image 4D devant recevoir l'imagette résultante.

Les paramètres optionnels *largeur* et *hauteur* vous permettent de définir la taille en pixels de l'imagette que vous souhaitez obtenir. Si vous omettez ces paramètres, la taille par défaut de l'imagette sera de 48x48 pixels.

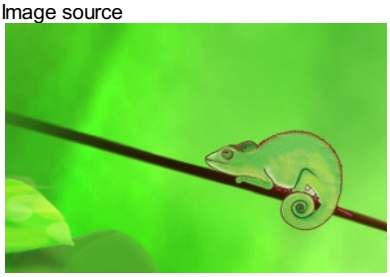
Le paramètre optionnel *mode* vous permet de définir le mode de création de l'imagette, c'est-à-dire la manière dont elle sera redimensionnée. Vous pouvez utiliser l'un des trois modes suivants, accessibles par l'intermédiaire de constantes prédéfinies disponibles dans le thème "**Formats d'affichage des images**" :

Constante	Type	Valeur
Non tronquée	Entier long	2
Proportionnelle	Entier long	5
Proportionnelle centrée	Entier long	6

Note : Seules ces trois constantes peuvent être utilisées avec **CREER IMAGETTE**. Les autres constantes du thème "Formats d'affichage des images" ne s'appliquent pas à cette commande.

Si vous omettez le paramètre *mode*, le mode 6 (Proportionnelle centrée) est appliqué par défaut. Le résultat des différents modes est illustré ci-dessous :

Image source



Imagettes résultantes (48x48)

- Non tronquée = 2

- Proportionnelle = 5

- Proportionnelle centrée = 6 (mode par défaut)

Note : Avec les modes "Proportionnelle" et "Proportionnelle centrée", les espaces vides apparaîtront blancs dans les imagettes — lorsque ces modes sont appliqués aux champs ou variables images dans les formulaires 4D, les espaces vides sont transparents.

Le paramètre *profondeur* est ignoré et doit être omis. La commande utilise toujours la profondeur écran (nombre de couleurs) courante.

ECRIRE FICHIER IMAGE

ECRIRE FICHIER IMAGE (nomFichier ; image {; codec})

Paramètre	Type		Description
nomFichier	Alpha	⇒	Nom ou chemin d'accès complet du fichier à écrire, ou chaîne vide
image	Image	⇒	Champ ou variable image à écrire
codec	Chaîne	⇒	Identifiant de codec d'image

Description

La commande **ECRIRE FICHIER IMAGE** vous permet de sauvegarder dans un fichier sur disque l'image passée dans le paramètre *image*, au format défini par *codec*.

Vous pouvez passer dans *nomFichier* le chemin d'accès complet du fichier à créer, ou uniquement le nom du fichier — auquel cas le fichier sera créé à côté du fichier de structure de la base. Vous devez également passer l'extension du fichier à créer.
Si vous passez une chaîne vide ("") dans *nomFichier*, la boîte de dialogue standard d'enregistrement de fichier apparaît, permettant à l'utilisateur de désigner le nom, l'emplacement et le format du fichier à créer. Si un nom par défaut a été associé au champ image, il est proposé dans la boîte de dialogue (cf. commande **FIXER NOM FICHIER IMAGE**).

Passez dans *image* la variable ou le champ image contenant l'image à stocker sur le disque.

Le paramètre optionnel *codec* vous permet de définir le format dans lequel l'image doit être sauvegardée. Un codec peut être une extension (par exemple ".gif") ou un type Mime (par exemple "image/jpg"). Vous pouvez obtenir la liste des codecs disponibles via la commande **LISTE CODECS IMAGES**.

Si vous omettez le paramètre *codec*, la commande tentera de déterminer le codec sur la base de l'extension du nom de fichier passé dans le paramètre *nomFichier*. Par exemple, si vous passez l'instruction :

```
ECRIRE FICHIER IMAGE("c:\dossier\photo.jpg";maphoto)
```

... la commande utilisera le codec JPEG pour stocker l'image.
Si l'extension utilisée ne correspond à aucun codec disponible, le fichier n'est pas enregistré et la variable système OK prend la valeur 0. Si vous ne passez ni *codec* ni extension de fichier, le fichier image est enregistré au format PICT.

Note : Si le format d'écriture de l'*image* (indiqué via l'extension de *nomFichier* ou le paramètre *codec*) est égal à son type d'origine et si aucune opération de transformation ne lui a été appliquée, l'image est écrite telle quelle, sans aucune modification.

Si l'exécution de la commande est correcte, la variable système Document contient le chemin d'accès complet du fichier créé et la variable système OK prend la valeur 1. En cas d'échec, OK prend la valeur 0.

ECRIRE IMAGE DANS BIBLIOTHEQUE

ECRIRE IMAGE DANS BIBLIOTHEQUE (image ; refImage ; nomImage)

Paramètre	Type		Description
image	Image	➡	Nouvelle image
refImage	Entier long	➡	Numéro de référence de l'image dans la bibliothèque d'images
nomImage	Chaîne	➡	Nouveau nom de l'image

Description

La commande **ECRIRE IMAGE DANS BIBLIOTHEQUE** crée une nouvelle image ou remplace une image existante dans la bibliothèque d'images.

Avant l'appel, vous passez :

- le numéro de référence de l'image dans *refImage* (compris entre 1 et 32767)
- l'image elle-même dans *image*.
- Le nom de l'image dans *nomImage* (longueur maximale : 255 caractères).

S'il existe déjà dans la bibliothèque une image possédant le même numéro de référence, son contenu est remplacé et elle est renommée avec les valeurs que vous avez passées dans *image* et *nomImage*.

Si aucune image ne possède le numéro de référence que vous avez passé dans *refImage*, une nouvelle image est créée dans la bibliothèque d'images.

4D Server : ECRIRE IMAGE DANS BIBLIOTHEQUE ne peut pas être utilisée dans une méthode exécutée sur le poste serveur (procédure stockée ou trigger). Si vous appelez **ECRIRE IMAGE DANS BIBLIOTHEQUE** sur le serveur, la commande ne fait rien, l'appel est ignoré.

Attention : Les objets de structure (éléments de listes hiérarchiques, lignes de menu, etc.) peuvent se référer à une image de la bibliothèque. Soyez prudent lorsque vous modifiez par programmation une image de la bibliothèque d'images.

Note : Si vous passez une image vide dans *image*, ou une valeur négative ou nulle dans *refImage*, la commande ne fait rien.

Exemple 1

Quel que soit le contenu courant de la bibliothèque d'images, l'exemple suivant ajoute une nouvelle image dans la bibliothèque en cherchant d'abord un numéro de référence d'image unique :

```
LISTE IMAGES DANS BIBLIOTHEQUE($alRefImage;$asNomImage)
Repeter
   $vlRefImage:=1+Abs(Hasard)
Jusque(Chercher dans tableau($alRefImage;$vlRefImage)<0)
ECRIRE IMAGE DANS BIBLIOTHEQUE(vgImage;$vlRefImage;"Nouvelle Image")
```

Exemple 2

L'exemple suivant importe dans la bibliothèque des images stockées dans un document sur disque, créé par le troisième exemple de la commande **LISTE IMAGES DANS BIBLIOTHEQUE** :

```
REGLER SERIE(10;"")
Si(OK=1)
   RECEVOIR VARIABLE($vsTag)
   Si($vsTag="4DV6BIBLIOTHEQUEIMAGEEXPORT")
      RECEVOIR VARIABLE($vlNbImages)
      Si($vlNbImages>0)
         Boucle($vlImage;1;$vlNbImages)
            RECEVOIR VARIABLE($vlRefImage)
            Si(OK=1)
               RECEVOIR VARIABLE($vsNomImage)
            Fin de si
            Si(OK=1)
               RECEVOIR VARIABLE($vgImage)
            Fin de si
            Si(OK=1)
               ECRIRE IMAGE DANS BIBLIOTHEQUE($vgImage;$vlRefImage;$vsNomImage)
            Sinon
               $vlImage:=$vlNbImages+1
               ALERTE("Ce fichier semble endommagé.")
            Fin de si
         Fin de boucle
      Sinon
         ALERTE("Ce fichier semble endommagé.")
      Fin de si
   Sinon
      ALERTE("Le fichier "+Document+" n'est pas un export de la bibliothèque d'images.")
   Fin de si
   REGLER SERIE(11)
Fin de si
```

Gestion des erreurs

S'il n'y a pas assez de mémoire pour retourner l'image, l'erreur –108 est générée. Notez que des erreurs d'E/S peuvent également être générées (si par exemple le fichier de structure est verrouillé). Vous pouvez intercepter ces erreurs avec une méthode de gestion d'erreurs.

Est un fichier image

Est un fichier image (cheminFichier {; *}) -> Résultat

Paramètre	Type		Description
cheminFichier	Texte	⇒	Chemin d'accès de fichier
*	Opérateur	⇒	Valider les données
Résultat	Booléen	⇐	Vrai = cheminFichier désigne un fichier image, sinon Faux

Description

La commande **Est un fichier image** teste le fichier désigné par le paramètre *cheminFichier* et retourne Vrai s'il s'agit d'un fichier image valide. La commande retourne Faux si le fichier n'est pas de type image ou s'il n'a pas été trouvé.

Passez dans le paramètre *cheminFichier* le chemin d'accès du fichier image à tester. Ce chemin doit être exprimé avec la syntaxe système. Vous pouvez passer un chemin d'accès absolu ou relatif au fichier de structure de la base. Si vous passez une chaîne vide (""), la commande retourne Faux.

Si vous ne passez pas le paramètre *, la commande teste le fichier en recherchant son extension parmi la liste des codecs disponibles. Si vous souhaitez pouvoir tester des fichiers sans extension ou effectuer une vérification plus complète, passez le paramètre *. Dans ce cas, la commande effectue des analyses supplémentaires : elle charge et inspecte l'en-tête du fichier et interroge les codecs afin de valider l'image. Cette syntaxe ralentit l'exécution de la commande.

Note : La commande retourne Vrai pour les fichiers PDF sous Windows et les fichiers EMF sous Mac OS.

FIXER METADONNEES IMAGE

FIXER METADONNEES IMAGE (image ; nomMeta ; contenuMeta {; nomMeta2 ; contenuMeta2 ; ... ; nomMetaN ; contenuMetaN})

Paramètre	Type		Description
image	Image	⇒	Image dont vous souhaitez écrire les métadonnées
nomMeta	Texte	⇒	Nom ou chemin du bloc à écrire
contenuMeta	Variable	⇒	Contenu de la métadonnée

Description

La commande **FIXER METADONNEES IMAGE** permet d'écrire ou de modifier le contenu de métadonnées (ou méta-balises) présentes dans *image* (champ ou une variable image 4D), lorsqu'elles sont modifiables.

Les métadonnées sont des informations supplémentaires insérées dans les images. 4D permet de manipuler quatre types de métadonnées standard : EXIF, GPS, IPTC et TIFF.

Note : Pour une description détaillée de ces types de metadonnées, vous pouvez consulter les documents suivants : http://www.iptc.org/std/IIM/4.1/specification/IIMV4.1.pdf (IPTC) et http://exif.org/Exif2-2.PDF (TIFF, EXIF et GPS).

Passez dans le paramètre *nomMeta* une chaîne désignant le type de métadonnée à écrire ou à modifier. Vous pouvez passer :

- une des constantes du thème **Noms des métadonnées images**. Ce thème regroupe toutes les balises prises en charge par 4D. Chaque constante contient un chemin de balise (par exemple "TIFF/DateTime"),
- le nom d'un bloc complet de métadonnées ("TIFF", "EXIF", "GPS" ou "IPTC"),
- une chaîne vide ("").

Passez dans le paramètre *contenuMeta* les nouvelles valeurs de la métadonnée :

- Si vous avez passé une constante de chemin de balise dans *nomMeta*, passez directement dans *contenuMeta* la valeur à écrire ou l'une des constantes appropriées du thème **Valeurs des métadonnées images**. La valeur peut être de type texte, entier long, réel, date ou heure, en fonction de la métadonnée désignée. Vous pouvez utiliser un tableau si la métadonnée contient plus d'une valeur. Si vous passez une chaîne, elle doit être formatée en XML (norme XMP). Passez une chaîne vide ("") pour effacer la métadonnée si elle existe.
- Si vous avez passé un nom de bloc ou une chaîne vide dans *nomMeta*, passez dans *contenuMeta* la référence XML DOM de l'élément contenant les métadonnées à écrire. Dans le cas d'une chaîne vide, toutes les métadonnées seront modifiées.

Attention : Certaines métadonnées sont en lecture seulement, par exemple TIFF xResolution/TIFF yResolution, EXIF color space ou EXIF pixel X dimension/EXIF pixel Y dimension, elles ne peuvent donc pas être modifiées par la commande **FIXER METADONNEES IMAGE**.

Sous Windows, si une erreur se produit durant l'exécution de la commande, la variable OK prend la valeur 0. A noter que sous Mac OS, pour des raisons techniques, les erreurs d'écriture des métadonnées ne sont pas détectées. La variable OK n'est pas modifiée par cette commande sous Mac OS.

Notes :

- Seuls certains formats d'images (notamment JPEG et TIFF) prennent en charge les métadonnées. A l'inverse, des formats tels que GIF ou BMP n'acceptent pas les métadonnées. En cas de conversion d'une image avec métadonnées dans un format ne les prenant pas en charge, les informations sont perdues.
- Sous OS X version 10.7 (Lion), un bogue du framework natif utilisé pour l'encodage et le décodage des métadonnées d'images peut entraîner des erreurs de précision dans les coordonnées GPS. Dans ce cas, une mise à jour vers OS X 10.8 (Mountain Lion) ou 10.9 (Maverick) est fortement recommandée.

Exemple 1

Ecriture de plusieurs valeurs de la métadonnée "Keywords" via des tableaux :

```
TABLEAU TEXTE($tTkeywords;2)
$tTkeywords{1}:="france"
$tTkeywords{2}:="europe"
FIXER METADONNEES IMAGE(vImage;IPTC keywords;$tTkeywords)
```

Exemple 2

Ecriture du bloc GPS via une référence DOM :

```
C_TEXTE($domMetas)
$domMetas:=DOM Analyser source XML("metas.xml")
C_TEXTE($refGPS)
$refGPS:=DOM Chercher element XML($domMetas;"Metadatas/GPS")
Si(OK=1)
   FIXER METADONNEES IMAGE(vImage;"GPS";$refGPS)  //$refGPS pointe bien ici sur l'élement GPS
   ...
Fin de si
DOM FERMER XML($domMetas)
```

Note

Lorsque toutes les métadonnées sont manipulées via une référence d'éléments DOM, les balises sont stockées comme attributs attachés à un élément (enfant de l'élément référencé) dont le nom est le nom du bloc (TIFF, IPTC, etc.). Lorsqu'un bloc de métadonnées spécifique est manipulé, les balises du

bloc sont stockées comme attributs directement attachés à l'élement référencé par la commande.

FIXER NOM FICHIER IMAGE

FIXER NOM FICHIER IMAGE (image ; nomFichier)

Paramètre	Type		Description
image	Champ image, Variable image	⇒	Image dont vous souhaitez fixer le nom par défaut
nomFichier	Texte	⇒	Nom par défaut de l'image

Description

La commande **FIXER NOM FICHIER IMAGE** vous permet de définir ou de modifier le nom de fichier par défaut de l'image passée en paramètre.

Ce nom peut avoir été défini automatiquement à partir du nom d'origine du fichier image importé dans le champ ou la variable image, ou lors d'un appel précédent à **FIXER NOM FICHIER IMAGE**.

Le nom par défaut est utilisé comme nom de fichier en cas d'exportation de l'image dans un fichier disque. Si le contenu du champ est copié dans une variable ou dans un autre champ, le nom par défaut est également copié. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

IMAGE VERS BLOB

IMAGE VERS BLOB (image ; blobImage ; codec)

Paramètre	Type		Description
image	Image	⇨	Champ ou variable image
blobImage	BLOB	⇦	BLOB devant contenir l'image convertie
codec	Chaîne	⇨	Identifiant de codec d'image

Description

La commande **IMAGE VERS BLOB** convertit une image stockée dans une variable ou un champ 4D dans un autre format, et place l'image résultante dans un BLOB.

Vous passez dans le paramètre *image* une variable ou un champ 4D de type image et dans le paramètre *blobImage* la variable ou le champ BLOB devant contenir l'image convertie.

Vous passez dans le paramètre *codec* une chaîne indiquant le format de conversion souhaité.
Un codec peut être une extension (par exemple ".gif") ou un type Mime (par exemple "image/jpg"). Vous pouvez obtenir la liste des codecs disponibles via la commande **LISTE CODECS IMAGES**.

Après l'exécution de la commande, *blobImage* contient l'image au format souhaité.

Si la conversion s'est déroulée correctement, la variable système OK prend la valeur 1. Si la conversion échoue (convertisseur non disponible), OK prend la valeur 0 et le BLOB est généré vide (0 octet).

IMAGE VERS GIF

IMAGE VERS GIF (imagePICT ; blobGIF)

Paramètre	Type		Description
imagePICT	Image	⇒	Champ ou variable image
blobGIF	BLOB	⇐	BLOB contenant l'image de type GIF

Description

La commande **IMAGE VERS GIF** permet de créer une image au format GIF à partir d'une image (de type PICT) stockée dans une variable ou un champ 4D.

Vous passez dans le paramètre *imagePICT* une variable ou un champ 4D de type image, et dans le paramètre *blobGIF*, une variable ou un champ de type BLOB. Après l'exécution de la commande, *blobGIF* contient l'image au format GIF.

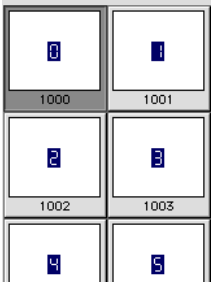
Note : Le format GIF est un format d'image comportant au plus 256 couleurs. Si l'image PICT d'origine en possède davantage, certaines couleurs seront perdues. La commande réduit le nombre de couleurs en fonction de la palette système. Le GIF généré est de type 87a (opaque) et normal (non entrelacé).

L'image incluse dans *blobGIF* pourra par la suite être enregistrée dans un fichier à l'aide de la commande **Windows Ctrl enfoncee** ou être utilisée en vue d'une publication sur le Web.

Si la conversion s'est déroulée correctement, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

Exemple

Vous souhaitez générer à la volée une image GIF affichant un compteur de connexions. Dans la bibliothèque d'images de la base, placez tous les chiffres sous forme d'images :



Dans la **Méthode base Sur connexion Web**, vous pouvez écrire :

```
  `Méthode base Sur connexion Web
Si(Contexte Web)
   ...
Sinon
   C_BLOB($blob)
   Au cas ou
         ...
      :($1="/4dcgi/counter") `Génération du compteur GIF
 `Lorsque 4D détecte cet URL lors de l'envoi de la page statique
         $blob:=gifcounter(◊nbHits) `Calcul de l'image gif
 `La variable ◊nbHits contient le nombre de connexions
         WEB ENVOYER BLOB($blob;"image/gif")
 `Insertion de l'image et envoi au navigateur
         ...
   Fin de cas
Fin de si
```

Voici la méthode ***gifcounter*** :

```
  `Méthode projet gifcounter
C_ENTIER LONG($1)
C_IMAGE($img)
C_BLOB($0)
Si($1=0)
   $ndigits:=1
Sinon
   $ndigits:=1+Longueur(Chaine($1))
Fin de si
Si($ndigits<5)
   $ndigits:=5
Fin de si
$div:=10^($ndigits-1)
Boucle($i;1;$ndigits)
   $ref:=Ent($1/$div)%10
   LIRE IMAGE DANS BIBLIOTHEQUE($ref+1000;picture)
   $img:=$img+picture
   $div:=$div/10
Fin de boucle
```

```
IMAGE VERS GIF($img;$0)
```

Lors de l'envoi de la page sur le navigateur, 4D affiche alors une image GIF du type suivant :

Variables et ensembles système

Si la conversion s'est déroulée correctement, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

Images egales

Images egales (image1 ; image2 ; masque) -> Résultat

Paramètre	Type		Description
image1	Champ image, Variable image	➡	Image source originale
image2	Champ image, Variable image	➡	Image à comparer
masque	Champ image, Variable image	⬅	Masque résultant
Résultat	Booléen	⮌	Vrai si les deux images sont identiques, sinon Faux

Description

La commande **Images egales** vous permet de comparer précisément deux images, tant au niveau de leurs dimensions que de leur contenu.

Passez dans *image1* l'image source et dans *image2* une image à comparer à l'image source.

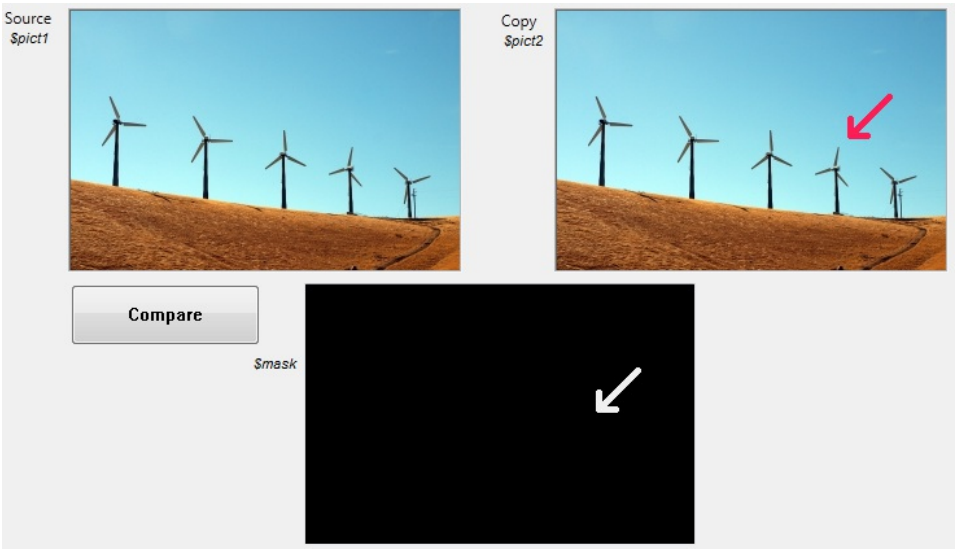
- Si les deux images sont de dimensions différentes, la commande retourne **Faux** et le paramètre *masque* contient une image vide.
- Si les deux images sont de même dimension mais ont des contenus différents, la commande retourne **Faux** et le paramètre *masque* contient l'image masque résultante de la comparaison des deux images. La comparaison est effectuée par pixel. Chaque pixel différent apparaît en blanc sur fond noir.
- Si les deux images sont identiques, la commande retourne **Vrai** et le paramètre *masque* contient une image noire.

Variables et ensembles système

La variable système OK prend la valeur 1 si les deux images ont pu être comparées. En cas d'anomalie, notamment si au moins une des deux images n'est pas initialisée (image vide), la variable OK prend la valeur 0.

Exemple

Dans l'exemple suivant, on compare deux images (pict1 et pict2) et on affiche le masque résultant :



Le code du bouton **Compare** est le suivant :

```
$equal :=Images egales($pict1;$pict2;$mask)
```

LIRE FICHIER IMAGE

LIRE FICHIER IMAGE (nomFichier ; image {; *})

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Nom ou chemin d'accès complet du fichier à lire, ou chaîne vide
image	Image	⇐	Champ ou variable recevant l'image
*	Opérateur	⇒	Si passé = accepter tout type de fichier

Description

La commande **LIRE FICHIER IMAGE** vous permet d'ouvrir l'image stockée dans le fichier disque désigné par *nomFichier* et de la placer dans le champ ou la variable 4D *image*.

Vous pouvez passer dans *nomFichier* le chemin d'accès complet du fichier à lire, ou uniquement le nom du fichier — auquel cas il doit se trouver à côté du fichier de structure de la base. Sous Windows, vous devez également passer l'extension du fichier.
Si vous passez une chaîne vide ("") dans *nomFichier*, la boîte de dialogue standard d'ouverture de documents apparaît, permettant à l'utilisateur de sélectionner le fichier à lire, ainsi que les formats disponibles.

Vous pouvez obtenir la liste des formats disponibles à l'aide de la commande **LISTE CODECS IMAGES**.

Passez dans *image* la variable ou le champ image devant recevoir l'image lue.

Note : Le format interne de l'image est conservé au sein de la variable ou du champ 4D.

Si vous passez le paramètre facultatif *, la commande acceptera tout type de fichier. Ce principe permet de manipuler des images sans nécessairement disposer des codecs adéquats (cf. description de la commande **BLOB VERS IMAGE**).

Variables et ensembles système

Si l'exécution de la commande est correcte, la variable système Document contient le chemin d'accès complet du fichier ouvert et la variable système OK prend la valeur 1. En cas d'échec, OK prend la valeur 0.

LIRE FORMATS IMAGE

LIRE FORMATS IMAGE (image ; tabCodecs)

Paramètre	Type		Description
image	Image	⇨	champ ou variable Image à analyser
tabCodecs	Tableau texte	⇦	Liste des codecs de l'image

Description

La commande **LIRE FORMATS IMAGE** remplit le tableau *tabCodecs* de tous les identifiants des codecs (formats image) de l'image contenue dans le paramètre *image*. Une image 4D (champ ou variable) peut contenir la même image encodée dans différents formats, comme PNG, BMP, GIF, etc.

Dans le paramètre *image*, vous passez un champ image ou une variable image qui inclut les formats que vous souhaitez récupérer dans le tableau *tabCodecs*.

Les identifiants des codecs sont établis par 4D exactement de la même façon qu'avec la commande **LISTE CODECS IMAGES**. Ils peuvent prendre les formes suivantes :

- une extension (par exemple, ".gif")
- un type Mime (par exemple, "image/jpeg")
- un code QuickTime sur 4 caractères

Notes:

- Les codecs suivants, gérés par 4D, sont toujours retournés en tant qu'extensions : JPEG, PNG, TIFF, GIF, BMP, SVG, PDF, EMF.
- Les codes QuickTime sur 4 caractères peuvent être retournés dans les bases de données où l'option de compatibilité QuickTime a été mise en place (en utilisant la commande **FIXER PARAMETRE BASE**). Toutefois, notez que QuickTime n'est plus pris en charge par 4D et que nous en déconseillons l'utilisation.

Pour plus d'information sur les identifiants des Codecs des images, référez-vous à la section **Introduction aux images**.

Exemple

Vous souhaitez connaître les formats de l'image stockée dans un champ Image de l'enregistrement courant :

```
TABLEAU TEXTE($aTPictureFormats;0)
 //lire tous les formats de l'image
LIRE FORMATS IMAGE([Employees]Photo;$aTPictureFormats)
```

LIRE IMAGE DANS BIBLIOTHEQUE

LIRE IMAGE DANS BIBLIOTHEQUE (refImage | nomImage ; image)

Paramètre	Type		Description
refImage | nomImage	Entier long, Chaîne	⇨	Numéro de référence ou Nom d'une image de la bibliothèque d'images
image	Variable image	⇦	Image de la bibliothèque d'images

Description

La commande **LIRE IMAGE DANS BIBLIOTHEQUE** retourne dans *image* l'image de la bibliothèque dont vous avez passé le numéro de référence dans *refImage* ou le nom dans *nomImage*.
S'il n'existe pas d'image de ce numéro ou de ce nom dans la bibliothèque d'images, **LIRE IMAGE DANS BIBLIOTHEQUE** ne modifie pas le paramètre *image*.

Exemple 1

L'exemple suivant retourne dans la variable *vgMonImage* l'image dont la référence est stockée dans la variable locale *$vlRefImage* :

```
LIRE IMAGE DANS BIBLIOTHEQUE($vlRefImage;vgMonImage)
```

Exemple 2

L'exemple suivant retourne dans la variable *$DDcom_Prot_MonImage* l'image nommée "DDcom_Prot_Bouton1" stockée dans la Bibliothèque d'images :

```
LIRE IMAGE DANS BIBLIOTHEQUE("DDcom_Prot_Bouton1";$DDcom_Prot_MonImage)
```

Exemple 3

Reportez-vous au troisième exemple de la commande **LISTE IMAGES DANS BIBLIOTHEQUE**.

Variables et ensembles système

La variable système OK prend la valeur 1 si l'image existe dans la bibliothèque d'images. Sinon, elle prend la valeur zéro.

Gestion des erreurs

S'il n'y a pas assez de mémoire pour retourner l'image, l'erreur –108 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs.

LIRE METADONNEES IMAGE

LIRE METADONNEES IMAGE (image ; nomMeta ; contenuMeta {; nomMeta2 ; contenuMeta2 ; ... ; nomMetaN ; contenuMetaN})

Paramètre	Type		Description
image	Image	⇒	Image dont vous souhaitez lire les métadonnées
nomMeta	Texte	⇒	Nom ou chemin du bloc à lire
contenuMeta	Variable	⇐	Contenu de la métadonnée

Description

La commande **LIRE METADONNEES IMAGE** permet de lire le contenu de métadonnées (ou méta-balises) présentes dans *image* (champ ou une variable image 4D). Pour plus d'informations sur les métadonnées, reportez-vous à la description de la commande **FIXER METADONNEES IMAGE**.

Passez dans le paramètre *nomMeta* une chaîne désignant le type de métadonnée à récupérer. Vous pouvez passer :

- une constante du thème "**Noms des métadonnées images**" contenant un chemin de balise,
- le nom d'un bloc complet de métadonnées ("TIFF", "EXIF", "GPS" ou "IPTC"),
- une chaîne vide ("").

Passez dans le paramètre *contenuMeta* la variable destinée à recevoir les métadonnées.

- Si vous avez passé un chemin de balise dans *nomMeta*, *contenuMeta* contient directement la valeur à lire. La valeur sera convertie dans le type de la variable. Les variables de type texte seront formatées en XML (norme XMP). Vous pouvez passer un tableau lorsque la métadonnée contient plus d'une valeur (c'est le cas notamment pour les balises IPTC keywords).
- Si vous avez passé un nom de bloc ou une chaîne vide dans *nomMeta*, *contenuMeta* doit être une référence d'élément DOM XML valide. Dans ce cas, le contenu du bloc désigné (ou de tous les blocs si vous avez passé une chaîne vide dans *nomMeta*) est recopié dans l'élément référencé.

Exemple 1

Utilisation d'arbres DOM

```
$xml:=DOM Creer ref XML("Root") //Création d'un arbre XML DOM

    //Réception des métadonnées TIFF
$_Xml_TIFF:=DOM Creer element XML($xml;"/Root/TIFF")
LIRE METADONNEES IMAGE(vPicture;"TIFF";$_Xml_TIFF)

    //Réception des métadonnées GPS
$_Xml_GPS:=DOM Creer element XML($xml;"/Root/GPS")
LIRE METADONNEES IMAGE(vPicture;"GPS";$_Xml_GPS)
```

Exemple 2

Utilisation de variables

```
C_DATE($dateAsDate)
LIRE METADONNEES IMAGE(vImage;TIFF date time;$dateAsDate)  //retourne uniquement la date car "$dateAsDate" est de type Date

C_TEXTE($dateAsText)
LIRE METADONNEES IMAGE(vImage;TIFF date time;$dateAsText)  //retourne uniquement la date mais au format XML

C_ENTIER($urgency)
LIRE METADONNEES IMAGE(vImage;IPTC urgency;$urgency)
```

Exemple 3

Réception de balises à valeurs multiples dans des tableaux

```
TABLEAU TEXTE($tTkeywords;0)
LIRE METADONNEES IMAGE(vImage;IPTC keywords;$tTkeywords)
```

Après exécution de la commande, tTkeywords contient par exemple :

```
$tTkeywords{1}="france"
$tTkeywords{2}="europe"
```

Exemple 4

Réception de balises à valeurs multiples dans une variable texte

```
C_TEXTE($vTmots;0)
LIRE METADONNEES IMAGE(vImage;IPTC keywords;$vTmots)
```

Après exécution de la commande, *vTmots* contient par exemple "france;europe".

Variables et ensembles système

La variable système OK retourne 1 si la récupération des métadonnées s'est bien passée, et 0 si une erreur se produit ou si au moins une des balises n'est pas trouvée. Dans tous les cas, les valeurs lisibles sont retournées.

LIRE MOTS CLES IMAGE

LIRE MOTS CLES IMAGE (image ; tabMotsclés {; *})

Paramètre	Type		Description
image	Champ image, Variable image	➡	Image dont vous souhaitez lire les mots-clés associés
tabMotsclés	Tableau texte	⬅	Tableau contenant les mots-clés extraits
*	Opérateur	➡	Si passé = utiliser les valeurs distinctes

Description

La commande **LIRE MOTS CLES IMAGE** retourne dans le tableau *tabMotsclés* la liste des mots-clés associés à l'image passée en paramètre.

Seuls les mots-clés définis via les métadonnées **IPTC/Keywords** sont pris en compte. Les autres types de métadonnées sont ignorés par la commande. La commande fonctionne uniquement avec les types d'images qui prennent en charge ce type de métadonnées (JPEG, TIFF...).

Note : Les métadonnées de type IPTC/Keywords sont indexables dans 4D (cf. manuel *Mode Développement*).

Si vous passez le paramètre *, la méthode ne retourne que les "valeurs disctinctes" de mots-clés, c'est-à-dire une liste sans doublons.

Si l'image ne contient pas de mots-clés ou de métadonnées IPTC/Keywords, la commande retourne un tableau vide, aucune erreur n'est générée.

Note : Les résultats retournés par cette commande peuvent différer en fonction de la valeur courante de la propriété de la base "N'utiliser que les caractères non alphanumériques pour les mots clés" (cf. paragraphe **Page Base de données/Stockage des données**).

Lire nom fichier image

Lire nom fichier image (image) -> Résultat

Paramètre	Type		Description
image	Champ image, Variable image	⇒	Image dont vous souhaitez obtenir le nom par défaut
Résultat	Texte	⇐	Nom par défaut du fichier image

Description

La commande **Lire nom fichier image** retourne le nom par défaut courant de l'image passée en paramètre.

Le nom par défaut est utilisé lors de l'exportation de l'image dans un fichier disque. Il peut être défini automatiquement à partir du nom d'origine du fichier image importé dans le champ ou la variable image, ou à l'aide de la commande **FIXER NOM FICHIER IMAGE**. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Si l'image n'a pas de nom par défaut, la commande retourne une chaîne vide.

LISTE CODECS IMAGES

LISTE CODECS IMAGES (tabCodecs {; tabNoms}{; *})

Paramètre	Type		Description
tabCodecs	Tableau chaîne	⇐	Identifiants des codecs d'images disponibles
tabNoms	Tableau chaîne	⇐	Noms des codecs d'images
*	Opérateur	⇒	Retourner la liste des codecs de lecture

Description

La commande **LISTE CODECS IMAGES** remplit le tableau *tabCodecs* avec la liste des identifiants des codecs d'images disponibles sur la machine où elle est exécutée. Cette liste comporte les codecs des formats d'images gérés en natif par 4D.

Les identifiants des codecs peuvent être retournés dans le tableau *tabCodecs* sous les formes suivantes :

- une extension (par exemple ".gif")
- un type Mime (par exemple "image/jpg")

Note de compatibilité : Si Quicktime a été activé dans la base (cf. section **Introduction aux images**), des codes QuickTime sur 4 caractères peuvent également être retournés (par exemple "PNTG").

La forme renvoyée par la commande dépend du mode de déclaration du codec au niveau du système d'exploitation. Le tableau facultatif *tabNoms* permet de récupérer le nom de chaque codec. Ces noms sont plus explicites que les identifiants. Ce tableau permet par exemple de construire et d'afficher un menu listant les codecs disponibles.

Par défaut, si vous ne passez pas le paramètre *, la commande retourne uniquement les codecs utilisables pour encoder (écrire) les images. Ces identifiants peuvent être utilisés dans le paramètre *format* des commandes d'exportation d'images **ECRIRE FICHIER IMAGE** et **IMAGE VERS BLOB**.
Si vous passez le paramètre *, la commande retourne également la liste des codecs utilisables pour décoder (lire) les images. Les deux listes ne sont pas exclusives, certains codecs de lecture et d'écriture sont identiques. Les codecs destinés à l'encodage des images pourront généralement être utilisés pour le décodage. En revanche, les codecs de décodage ne permettent pas forcément l'encodage. Par exemple, le codec ".jpg" sera présent dans les deux listes, tandis que le codec ".xbmp" sera présent dans la liste des codecs de lecture mais dans celle d'écriture.

LISTE IMAGES DANS BIBLIOTHEQUE

LISTE IMAGES DANS BIBLIOTHEQUE (refsImages ; nomsImages)

Paramètre	Type		Description
refsImages	Tableau entier long	⇦	Numéros de référence des images stockées dans la bibliothèque d'images
nomsImages	Tableau chaîne	⇦	Noms des images stockées dans la bibliothèque d'images

Description

La commande **LISTE IMAGES DANS BIBLIOTHEQUE** retourne les numéros de référence et le nom des images stockées dans la bibliothèque d'images de la base de données.

Après l'appel, vous récupérez les numéros de référence des images dans le tableau *refsImages* et leurs noms dans le tableau *nomsImages*. Les deux tableaux sont synchronisés : le nième élément de *refsImages* est le numéro de référence de l'image de la bibliothèque dont le nom est retourné dans le nième élément de *nomsImages*.

Si nécessaire, la commande crée et dimensionne automatiquement les tableaux *refsImages* et *nomsImages*.

La longueur maximale du nom d'une image de la bibliothèque est de 255 caractères.

Si la bibliothèque d'images est vide, les deux tableaux retournés seront vides.

Pour obtenir le nombre d'images contenues dans la bibliothèque, il vous suffit de tester la taille d'un des deux tableaux à l'aide de la fonction **Taille tableau**.

Exemple 1

Le code suivant retourne le contenu de la bibliothèque d'images dans les tableaux *telRefImage* et *taNomImage* :

```
LISTE IMAGES DANS BIBLIOTHEQUE(telRefImage;taNomImage)
```

Exemple 2

L'exemple suivant teste si la bibliothèque d'images est vide ou non :

```
LISTE IMAGES DANS BIBLIOTHEQUE(telRefImage;taNomImage)
Si(Taille tableau(telRefImage)=0)
   ALERTE("La bibliothèque d'images est vide.")
Sinon
   ALERTE("La bibliothèque d'images contient "+Chaine(Taille tableau(tlRefImage))+" images.")
Fin de si
```

Exemple 3

L'exemple suivant exporte la Bibliothèque d'Images vers un document stocké sur disque :

```
LISTE IMAGES DANS BIBLIOTHEQUE($alRefImage;$asNomImage)
$vlNbImages:=Taille tableau($alRefImage)
Si($vlNbImages>0)
   REGLER SERIE(12;"")
   Si(OK=1)
      $vsTag:="4DV6PICTURELIBRARYEXPORT"
      ENVOYER VARIABLE($vsTag)
      ENVOYER VARIABLE($vlNbImages)
      gError:=0
      Boucle($vlImage;1;$vlNbImages)
         $vlRefImage:=$alRefImage{$vlImage}
         $vsNomImage:=$asNomImage{$vlImage}
         LIRE IMAGE DANS BIBLIOTHEQUE($alRefImage{$vlImage};$vgImage)
         Si(OK=1)
            ENVOYER VARIABLE($vlRefImage)
            ENVOYER VARIABLE($vsNomImage)
            ENVOYER VARIABLE($vgImage)
         Sinon
            $vlImage:=$vlImage+1
            gError:=-108
         Fin de si
      Fin de boucle
      REGLER SERIE(11)
      Si(gError#0)
         ALERTE("La bibliothèque d'images n'a pas pu être exportée, recommencez avec davantage de mémoire.")
         SUPPRIMER DOCUMENT(Document)
      Fin de si
   Fin de si
Sinon
   ALERTE("La bibliothèque d'images est vide.")
Fin de si
```

PROPRIETES IMAGE

PROPRIETES IMAGE (image ; largeur ; hauteur {; hOffset {; vOffset {; mode}}})

Paramètre	Type		Description
image	Image	⇒	Image sur laquelle obtenir les informations
largeur	Entier long	⇐	Largeur de l'image exprimée en pixels
hauteur	Entier long	⇐	Hauteur de l'image exprimée en pixels
hOffset	Entier long	⇐	Offset horizontal lorsque l'image est affichée en arrière-plan
vOffset	Entier long	⇐	Offset vertical lorsque l'image est affichée en arrière-plan
mode	Entier long	⇐	Mode de transfert lorsque l'image est affichée en arrière-plan

Description

La commande **PROPRIETES IMAGE** retourne des informations sur l'image que vous avez passée dans le paramètre *image*.

Les paramètres *largeur* et *hauteur* reçoivent la largeur et hauteur réelles de l'image.

Les paramètres *hOffset*, *vOffset* et *mode* reçoivent la position et le mode de transfert de l'image lorsqu'elle est affichée en arrière-plan dans un formulaire (“Image sur fond”).

SUPPRIMER IMAGE DANS BIBLIOTHEQUE

SUPPRIMER IMAGE DANS BIBLIOTHEQUE (refImage | nomImage)

Paramètre	Type		Description
refImage | nomImage	Entier long, Chaîne	⇨	Numéro de référence ou Nom d'une image de la bibliothèque d'images

Description

La commande **SUPPRIMER IMAGE DANS BIBLIOTHEQUE** supprime de la bibliothèque d'images l'image dont vous avez passé le numéro de référence dans *refImage* ou le nom dans *nomImage*.
Si ce numéro de référence ou ce nom ne correspond à aucune image, la commande ne fait rien.

4D Server : SUPPRIMER IMAGE DANS BIBLIOTHEQUE ne peut pas être utilisée dans une méthode exécutée sur le poste serveur (procédure stockée ou trigger). Si vous appelez **SUPPRIMER IMAGE DANS BIBLIOTHEQUE** sur le serveur, il ne se passe rien, l'appel est ignoré.

Attention : Les objets de structure (éléments de liste hiérarchique, lignes de menu, etc.) peuvent se référer à une image de la bibliothèque. Soyez prudent lorsque vous supprimez par programmation une image de la bibliothèque d'images.

Exemple 1

L'exemple suivant supprime l'image n°4444 de la bibliothèque d'images :

```
SUPPRIMER IMAGE DANS BIBLIOTHEQUE(4444)
```

Exemple 2

L'exemple suivant supprime de la bibliothèque d'images celles dont le nom commence par le symbole dollar ($) :

```
LISTE IMAGES DANS BIBLIOTHEQUE($alRefImage;$asNomImage)
Boucle($vlImage;1;Taille tableau($alRefImage))
   Si($asNomImage{$vlImage}="$@")
      SUPPRIMER IMAGE DANS BIBLIOTHEQUE($alRefImage{$vlImage})
   Fin de si
Fin de boucle
```

Taille image

Taille image (image) -> Résultat

Paramètre	Type		Description
image	Image	⇨	Image pour laquelle vous voulez connaître la taille en octets
Résultat	Entier long	⇦	Taille en octets de l'image

Description

Taille image retourne la taille de l'image *image* en octets.

TRANSFORMER IMAGE

TRANSFORMER IMAGE (image ; opérateur {; param1 {; param2 {; param3 {; param4}}}})

Paramètre	Type		Description
image	Image	⇒	Image source à transformer
		⇐	Image résultant de la transformation
opérateur	Entier long	⇒	Type de transformation à effectuer
param1	Réel	⇒	Paramètre de la transformation
param2	Réel	⇒	Paramètre de la transformation
param3	Réel	⇒	Paramètre de la transformation
param4	Réel	⇒	Paramètre de la transformation

Description

La commande **TRANSFORMER IMAGE** permet d'appliquer une transformation de type *opérateur* à l'image passée dans le paramètre *image*.

Note : Cette commande reprend et étend les fonctionnalités proposées par les opérateurs "classiques" de transformation d'images (+/, etc., cf. section **Opérateurs sur les images**). Ces opérateurs restent parfaitement utilisables dans 4D.

L'*image* source est modifiée directement à l'issue de l'exécution de la commande. A noter cependant que certaines opérations ne sont pas destructives et peuvent être annulées via l'opération inverse ou l'opération "Réinitialisation". Par exemple, une image réduite à 1 % retrouvera sa taille originale sans altération si elle est agrandie 100 fois par la suite. Les transformations ne modifient pas le type d'origine de l'image : par exemple, une image vectorielle restera vectorielle à l'issue de la transformation.

Passez dans *opérateur* le numéro de l'opération à effectuer et dans *param1* à *param4* le ou les paramètre(s) nécessaire(s) à cette opération (le nombre de paramètres dépend de l'opération). Vous pouvez utiliser dans *opérateur* l'une des constantes du thème "**Transformation des images**". Ces opérateurs et leurs paramètres sont décrits dans le tableau suivant :

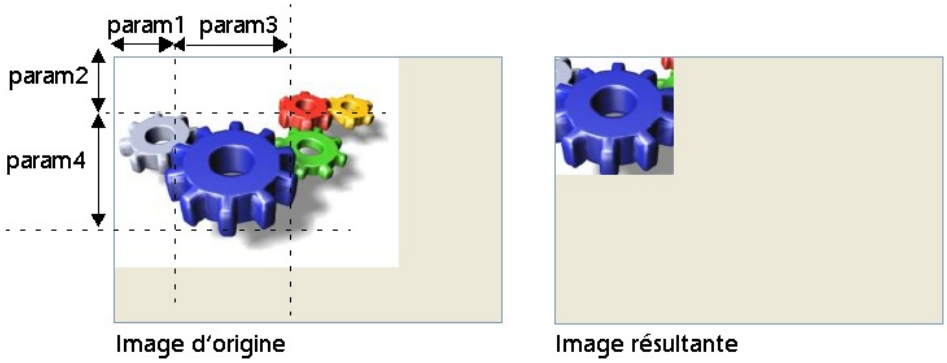
opérateur (valeur)	param1	param2	param3	param4	Valeurs	Annulable
Réinitialisation (0)	-	-	-	-	-	-
Redimensionnement (1)	Largeur	Hauteur	-	-	Facteurs	oui
Translation (2)	Axe X	Axe Y	-	-	Pixels	oui
Miroir horizontal (3)	-	-	-	-		oui
Miroir vertical (4)	-	-	-	-		oui
Recadrage (100)	Orig. X	Orig. Y	Largeur	Hauteur	Pixels	non
Passage en niveaux de gris (101)	-	-	-	-		non
Transparence (102)	Couleur RVB	-	-	-	Hexadécimal	non

- Réinitialisation : toutes les opérations matricielles effectuées sur l'image (redimensionnement, miroir...) sont annulées.
- Redimensionnement : l'image est redimensionnée horizontalement et verticalement en fonction des valeurs passées respectivement dans *param1* et *param2*. Ces valeurs sont des facteurs : par exemple, pour agrandir la largeur de 50 %, passez 1,5 dans *param1* et pour réduire la hauteur de 50 %, passez 0,5 dans *param2*.
- Translation : l'image est déplacée de *param1* pixels horizontalement et de *param2* pixels verticalement. Passez une valeur positive pour un déplacement vers la droite ou vers le bas et une valeur négative pour un déplacement vers la gauche ou vers le haut.
- Miroir horizontal et Miroir vertical : l'effet miroir est appliqué à l'image d'origine. Tout déplacement éventuel effectué auparavant ne sera pas pris en compte.
- Recadrage : l'image est recadrée à partir du point de coordonnées *param1* et *param2* (exprimé en pixels). La largeur et la hauteur de la nouvelle image sont déterminées par les paramètres *param3* et *param4*. Cette transformation ne peut pas être annulée.
- Passage en niveaux de gris : l'image est passée en niveaux de gris (aucun paramètre n'est requis). Cette transformation ne peut pas être annulée.
- Transparence : Un masque de transparence est appliqué à l'image sur la base de la couleur passée dans *param1*. Par exemple, si vous passez 0x00FFFFFF (blanc) dans *param1*, tous les pixels blancs de l'image originale seront transparents dans l'image transformée. Cette opération peut être appliquée aux images bitmap ou vectorielles. Par défaut, si le paramètre *param1* est omis, le blanc (0x00FFFFFF) sera utilisé comme couleur cible. Cette fonction est plus particulièrement destinée à gérer la transparence dans les images converties depuis le format obsolète PICT, mais peut être utilisée avec des images de tout type. Cette transformation ne peut pas être annulée.

Exemple 1

Voici un exemple de recadrage (l'image est affichée dans le formulaire avec le format "Image tronquée (non centrée)") :

```
TRANSFORMER IMAGE($vpRouages;Recadrage;50;50;100;100)
```

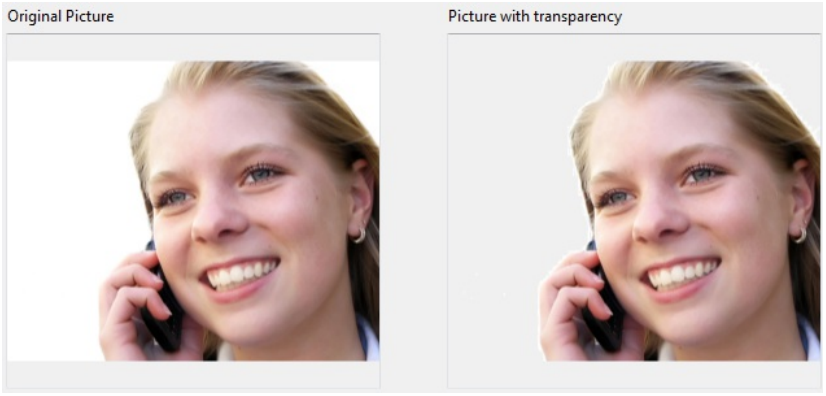


Exemple 2

Vous souhaitez transformer les parties blanches d'une image en parties transparentes. Pour cela, vous pouvez utiliser le code suivant :

```
TRANSFORMER IMAGE(Pict1;Transparence;0x00FFFFFF) //0x00FFFFFF est le blanc
```

Vous obtenez le résultat suivant :



_o_ENREGISTRER IMAGE

_o_ENREGISTRER IMAGE (document ; image)

Paramètre	Type		Description
document	RefDoc	⇨	Numéro de référence du document
image	Image	⇨	Image à enregistrer

Note de compatibilité

Cette commande fait appel à des mécanismes obsolètes et est conservée pour des raisons de compatibilité uniquement. Elle est avantageusement remplacée par la commande **ECRIRE FICHIER IMAGE**.

_o_LISTE TYPES IMAGES

_o_LISTE TYPES IMAGES (tabFormats {; tabLibellés})

Paramètre	Type		Description
tabFormats	Tableau chaîne	⇐	Codes QuickTime des formats d'import/export disponibles
tabLibellés	Tableau chaîne	⇐	Noms des formats

Note de compatibilité

Cette commande nécessite la présence de QuickTime et ne donne pas accès aux formats gérés en natif par 4D depuis la version 11. Son intérêt est désormais limité et elle est avantageusement remplacée par la commande **LISTE CODECS IMAGES**.

_o_QT CHARGER ET COMPRESSER IMAGE

_o_QT CHARGER ET COMPRESSER IMAGE (document ; méthode ; qualité ; image)

Paramètre	Type		Description
document	RefDoc	⇨	Numéro de référence du document
méthode	Chaîne	⇨	Type de compression
qualité	Entier long	⇨	Qualité de compression (1...1000)
image	Image	⇦	Image compressée

Note de compatibilité

Cette commande fait appel à des mécanismes obsolètes et doit désormais être remplacée par les commandes **LIRE FICHIER IMAGE** et **CONVERTIR IMAGE**.

_o_QT COMPRESSER FICHIER IMAGE

_o_QT COMPRESSER FICHIER IMAGE (document ; méthode ; qualité)

Paramètre	Type		Description
document	RefDoc	⇒	Numéro de référence du document
méthode	Chaîne	⇒	Type de compression
qualité	Entier long	⇒	Qualité de compression (1...1000)

Note de compatibilité

Cette commande fait appel à des mécanismes obsolètes et doit désormais être remplacée par les commandes **ECRIRE FICHIER IMAGE** ou **IMAGE VERS BLOB**.

_o_QT COMPRESSER IMAGE

_o_QT COMPRESSER IMAGE (image ; méthode ; qualité)

Paramètre	Type		Description
image	Image	⇒	Image à compresser
		⇐	Image compressée
méthode	Chaîne	⇒	Type de compression (4 caractères)
qualité	Entier long	⇒	Qualité de compression (1...1000)

Note de compatibilité

Cette commande fait appel à des mécanismes obsolètes et doit désormais être remplacée par la commande **CONVERTIR IMAGE**.

Import-Export

- ECRITURE DIF
- ECRITURE SYLK
- EXPORTER DONNEES
- EXPORTER ODBC
- EXPORTER TEXTE
- IMPORTER DONNEES
- IMPORTER ODBC
- IMPORTER TEXTE
- LECTURE DIF
- LECTURE SYLK

ECRITURE DIF

ECRITURE DIF ({laTable ;} nomFichier)

Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle effectuer l'export ou Table par défaut si ce paramètre est omis
nomFichier	Chaîne	⇒	Document DIF à exporter

Description

La commande **ECRITURE DIF** écrit dans *document* (document DIF Windows ou Mac OS) les données des enregistrements de la sélection courante de la table *laTable* du process courant.

L'opération d'export s'effectue par l'intermédiaire du formulaire sortie courant. L'opération d'export écrit les champs et les variables en fonction de l'ordre de saisie dans le formulaire sortie. C'est pourquoi vous devez veiller à utiliser un formulaire sortie ne contenant que les champs ou objets saisissables que vous voulez exporter. Ne placez pas de boutons ou autres objets sur le formulaire sortie. Les objets de sous-formulaire sont ignorés.

L'événement Sur chargement est envoyé à la méthode du formulaire pour chaque enregistrement exporté. Utilisez cet événement pour définir les variables utilisées dans le formulaire d'export.

Le paramètre *document* peut créer un nouveau document ou désigner un document existant. Si le document a le même nom qu'un document existant, ce dernier est remplacé. Le document peut contenir un chemin d'accès de volume et/ou de dossiers. Si vous passez une chaîne vide, la boîte de dialogue standard de sauvegarde de fichiers est affichée. Si l'utilisateur annule ce dialogue, l'opération d'export est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'export. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Si l'export s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande **SUPPRIMER MESSAGES**.

La commande utilise par défaut le jeu de caractères UTF-8. Les documents au format DIF utilisant généralement le jeu de caractères IBM437, il peut être nécessaire d'utiliser la commande **UTILISER FILTRE** pour définir le jeu de caractères approprié.

Lors de l'utilisation de **ECRITURE DIF**, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrement est par défaut le retour chariot (code 13). Vous pouvez modifier ces valeurs en assignant de nouvelles valeurs aux deux **Variables système** *FldDelimit* et *RecDelimit*. L'utilisateur peut modifier ces valeurs par défaut dans le dialogue d'export du mode Développement. Comme les champs Texte peuvent contenir des retours chariot, soyez prudent si vous utilisez le retour chariot comme délimiteur entre les champs à exporter.

Exemple

Cet exemple exporte des données vers un document DIF. Cette méthode commence par le choix du formulaire sortie, puis effectue l'export :

```
FORM FIXER SORTIE([Personnes];"Export")
ECRITURE DIF([Personnes];"Nouvelles Personnes.dif") ` Export vers le document "Nouvelles Personnes.dif"
```

Variables et ensembles système

OK prend la valeur 1 si l'export s'est correctement déroulé, sinon elle prend la valeur 0.

ECRITURE SYLK

ECRITURE SYLK ({laTable ;} nomFichier)

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle effectuer l'export ou Table par défaut si ce paramètres est omis
nomFichier	Chaîne	⇨	Document SYLK à exporter

Description

La commande **ECRITURE SYLK** écrit dans *document* (document SYLK Windows ou Mac OS) les données des enregistrements de la sélection courante de la table *laTable* du process courant.

L'opération d'export s'effectue par l'intermédiaire du formulaire sortie courant. Les champs et les variables sont écrits en fonction de l'ordre de saisie dans le formulaire sortie. C'est pourquoi vous devez veiller à utiliser un formulaire sortie ne contenant que les champs ou objets saisissables que vous voulez exporter. Ne placez pas de boutons ou autres objets sur le formulaire sortie. Les objets de sous-formulaire sont ignorés.

L'événement Sur chargement est envoyé à la méthode du formulaire pour chaque enregistrement exporté. Utilisez cet événement pour définir les variables utilisées dans le formulaire d'export.

Le paramètre *document* peut créer un nouveau document ou désigner un document existant. Si le document a le même nom qu'un document existant, ce dernier est remplacé. Le document peut contenir un chemin d'accès de volume et/ou de dossiers. Si vous passez une chaîne vide, la boîte de dialogue standard de sauvegarde de fichiers est affichée. Si l'utilisateur annule ce dialogue, l'opération d'export est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'export. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Si l'export s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande **SUPPRIMER MESSAGES**.

La commande utilise par défaut le jeu de caractères UTF-8. Les documents au format SYLK utilisant généralement le jeu de caractères ISO-8859-1, il peut être nécessaire d'utiliser la commande **UTILISER FILTRE** pour fixer le jeu de caractères approprié.

Lors de l'utilisation de **ECRITURE SYLK**, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrement est par défaut le retour chariot (code 13) sous OS X et le retour chariot+retour à la ligne (code 13 + code 10) sous Windows. Vous pouvez modifier ces valeurs en assignant de nouvelles valeurs aux deux **Variables système** *FldDelimit* et *RecDelimit*. L'utilisateur peut modifier ces valeurs par défaut dans le dialogue d'export du mode Développement. A noter que si des champs exportés contiennent des caractères définis comme délimiteurs de champ ou d'enregistrement, ces caractères sont automatiquement remplacés par des espaces dans le fichier exporté, afin de ne pas perturber le processus d'import.

Exemple

Cet exemple exporte des données vers un document SYLK. La méthode commence par le choix du formulaire sortie, puis l'export est déclenché :

```
FORM FIXER SORTIE([Personnes];"Export")
ECRITURE SYLK([Personnes];"Nouvelles Personnes.slk") ` Export vers le document "Nouvelles Personnes.slk"
```

Variables et ensembles système

OK prend la valeur 1 si l'export s'est correctement déroulé, sinon elle prend la valeur 0.

EXPORTER DONNEES

EXPORTER DONNEES (nomFichier {; projet {; *}})

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Chemin d'accès et nom du fichier d'export
projet	Variable texte, Variable BLOB	⇒	Contenu du projet d'export (XML ou référence d'élément DOM ou BLOB)
		⇐	Nouveau contenu du projet d'export (si le paramètre * a été passé)
*	Opérateur	⇒	Affichage de la boîte de dialogue d'export et mise à jour du projet

Description

La commande **EXPORTER DONNEES** permet d'exporter des données dans le fichier *nomFichier*. 4D peut exporter des données au format Texte, Texte de longueur fixe, XML, SYLK, DIF, DBF (dBase), et 4D.

Si vous passez une chaîne vide dans le *nomFichier*, **EXPORTER DONNEES** provoque l'affichage d'une boîte de dialogue standard d'enregistrement de fichiers, permettant à l'utilisateur de définir le nom, le type et l'emplacement du fichier d'export. Une fois la boîte de dialogue validée, la variable système Document contient le chemin d'accès et le nom de ce fichier. Si l'utilisateur clique sur le bouton **Annuler**, l'exécution est stoppée et la variable système OK prend la valeur 0.

Le paramètre optionnel *projet* vous permet d'utiliser un projet pour l'export des données. Lorsque vous passez ce paramètre, l'export s'effectue directement, sans intervention de l'utilisateur (sauf si vous utilisez l'option *, cf. ci-dessous). Si vous ne passez pas ce paramètre, la boîte de dialogue de paramétrage d'export s'affiche, permettant à l'utilisateur de définir ses paramètres d'export ou de charger un projet d'export existant.

Un projet d'export contient tous les paramètres de l'export, tels que les tables et champs exportés, les délimiteurs, etc. Vous pouvez passer dans *projet* soit une variable texte contenant du XML, soit une variable texte contenant la référence à un élément DOM préexistant, soit un BLOB. Les projets peuvent avoir été créés par programmation (projets au format XML uniquement) ou être issus du chargement de paramètres préalablement définis dans la boîte de dialogue d'export. Dans ce dernier cas, vous disposez de deux solutions :

- utiliser la commande **EXPORTER DONNEES** avec un paramètre *projet* vide et le paramètre optionnel * (cf. ci-dessous), puis stocker le paramètre *projet* résultant dans un champ Texte ou BLOB. Cette solution permet notamment de conserver le projet avec le fichier de données.
- sauvegarder le projet sur disque, puis le charger par exemple à l'aide de la commande **DOM Analyser source XML** et passer sa référence dans le paramètre *projet*.

Note de compatibilité : A compter de la version 12 de 4D, les projets d'export sont encodés en XML. 4D peut ouvrir les projets d'export générés avec des versions précédentes de 4D (format BLOB), mais les projets créés à compter de la v12 ne peuvent plus être rouverts avec une v11 ou antérieure. Il est désormais conseillé d'utiliser des variables Texte pour manipuler les fichiers d'export.

Le paramètre optionnel *, s'il est est spécifié, provoque l'affichage de la boîte de dialogue de paramétrage d'export avec les paramétrages définis dans le projet. Ce fonctionnement permet d'utiliser un projet prédéfini, tout en ayant la possibilité de modifier un ou plusieurs paramètres. En outre, le paramètre projet contient, après la fermeture de la boîte de dialogue d'export, les paramètres du "nouveau" projet au format XML. Vous pouvez alors le stocker dans un champ Texte, dans un fichier sur disque, etc.

Si l'export se déroule correctement, la variable système OK prend la valeur 1.

Exemple 1

Cet exemple illustre l'utilisation de la commande **EXPORTER DONNEES** pour exporter en format binaire les données d'une base.

- Cette méthode effectue une boucle sur toutes les tables de la base et appelle la méthode ***ExportBinary*** :

```
C_TEXTE($ExportPath)
C_ENTIER LONG($i)
$ExportPath:=Selectionner dossier("Veuillez sélectionner le dossier d'export :")
Si(Ok=1)
   Boucle($i;1;Lire numero derniere table)
      Si(Est un numero de table valide($i))
         ExportBinary(Table($i);$ExportPath+Nom de la table($i);Vrai)
      Fin de si
   Fin de boucle
Fin de si
```

- Voici le code de la méthode ***ExportBinary*** :

```
C_POINTEUR($1) //table
C_TEXTE($2) //chemin du fichier de destination
C_BOOLEEN($3) //exporter tous les enregistrements
C_ENTIER LONG($i)
C_TEXTE($ref)
$ref:=DOM Creer ref XML("settings-import-export")
 // Exporter la table "$1" au format binaire '4D', tous les enregistrements ou uniquement la sélection courante
DOM ECRIRE ATTRIBUT XML($ref;"table_no";Table($1);"format";"4D";"all_records";$3)
 // Définition des champs à exporter
Boucle($i;1;Lire numero dernier champ($1))
   Si(Est un numero de champ valide($1;$i))
      $elt:=DOM Creer element XML($ref;"field";"table_no";Table($1);"field_no";$i)
   Fin de si
Fin de boucle
EXPORTER DONNEES($2;$ref)
```

```
Si(Ok=0)
   ALERTE("Erreur lors de l'export de la table "+Nom de la table($1))
Fin de si
DOM FERMER XML($ref)
```

Exemple 2

Cet exemple crée un projet vide et y stockera les paramètres définis par l'utilisateur dans la boîte de dialogue d'export :

```
C_TEXTE($exportParams)
EXPORTER DONNEES("DocExport.txt";$exportParams;*) //Affichage de la boîte de dialogue d'export
```

Variables et ensembles système

Si l'utilisateur clique sur **Annuler** dans une des boîtes de dialogue (d'enregistrement de projet ou de paramétrage d'export), la variable système OK prend la valeur 0. Si l'export se déroule correctement, la variable système OK prend la valeur 1.

EXPORTER ODBC

EXPORTER ODBC (tableSource {; projet {; *}})

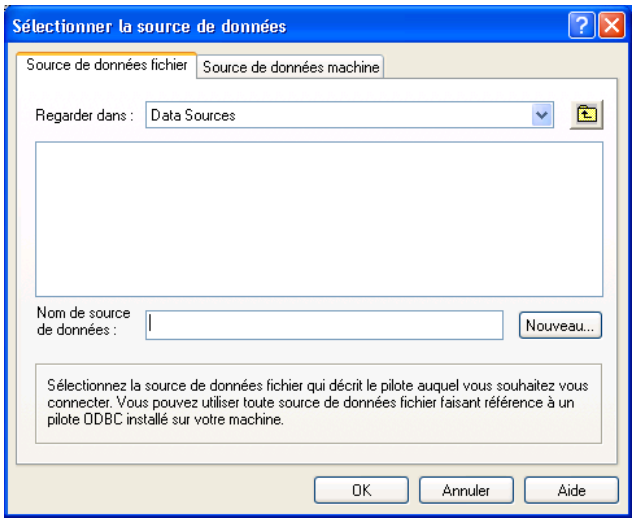
Paramètre	Type		Description
tableSource	Chaîne	➡	Nom de la table dans la source de données ODBC
projet	BLOB	➡	Contenu du projet d'exportation
		⬅	Nouveau contenu du projet (si * est passé)
*	Opérateur	➡	Affichage de la boîte de dialogue d'exportation et mise à jour du projet

Description

La commande **EXPORTER ODBC** permet d'exporter des données dans la table *tableSource* d'une source ODBC externe.

Si vous appelez la commande **EXPORTER ODBC** en-dehors d'une connexion préalablement ouverte à l'aide de la commande **SQL LOGIN**, la boîte de dialogue de sélection de la source de données ODBC est affichée :

Windows



Mac OS

Si l'utilisateur clique sur le bouton **Annuler** dans cette boîte de dialogue, l'exécution est stoppée et la variable système OK prend la valeur 0.

Note : Cette commande ne peut pas être utilisée dans le cadre d'une connexion avec le moteur SQL interne de 4D.

Si vous ne passez pas le paramètre *projet*, 4D affiche la boîte de dialogue d'exportation ODBC, permettant à l'utilisateur de configurer l'opération. Pour plus d'informations sur cette boîte de dialogue, reportez-vous au manuel Mode Développement.

Si vous passez dans le paramètre *projet* un BLOB contenant un projet d'export ODBC valide, l'exportation s'effectue directement, sans intervention de l'utilisateur. Pour cela, il vous suffit de charger un projet préalablement sauvegardé sur disque dans le champ ou la variable BLOB que vous passez dans le paramètre projet, à l'aide de la commande **DOCUMENT VERS BLOB**. Les projets d'export ODBC sont sauvegardés depuis la boîte de dialogue d'exportation ODBC.

Vous pouvez également utiliser la commande **EXPORTER ODBC** avec un paramètre *projet* vide et le paramètre optionnel *, puis stocker le paramètre *projet* dans un champ BLOB (cf. ci-dessous). Cette solution permet, d'une part, de le conserver avec le fichier de données, et d'autre part d'éviter la phase de chargement dans un BLOB depuis le disque.

Note : Reportez-vous à la commande **EXPORTER DONNEES** pour un exemple de définition de projet vide. A noter que les projets générés dans la boîte de dialogue d'exportation ODBC ne sont pas compatibles avec les commandes ou la boîte de dialogue d'exportation standard de 4D.

Le paramètre optionnel *, s'il est spécifié, provoque l'affichage de la boîte de dialogue de paramétrage d'exportation ODBC avec les paramétrages éventuellement définis dans le projet. Ce fonctionnement permet d'utiliser un projet prédéfini tout en ayant la possibilité de modifier un ou plusieurs paramètres. En outre, dans ce cas le paramètre *projet* contient, après la fermeture de la boîte de dialogue, les paramètres du "nouveau" projet. Vous pouvez alors le stocker dans un champ BLOB, sur disque, etc.

Variables et ensembles système

Si l'utilisateur clique sur le bouton **Annuler** dans une des boîtes de dialogue (de sélection de source de données ou de paramétrage de l'export), la variable système OK prend la valeur 0. Si l'exportation se déroule correctement, la variable système OK prend la valeur 1.

EXPORTER TEXTE

EXPORTER TEXTE ({laTable ;} nomFichier)

Paramètre	Type		Description
laTable	Table	⇒	Table depuis laquelle effectuer l'export ou Table par défaut si ce paramètre est omis
nomFichier	Chaîne	⇒	Document texte à exporter

Description

La commande **EXPORTER TEXTE** écrit dans *document* (document texte Windows ou Mac OS) les données des enregistrements de la sélection courante de la table *laTable* du process courant.

L'opération d'export s'effectue par le formulaire sortie courant. Les champs et les variables sont écrits en fonction de l'ordre de saisie dans le formulaire sortie. C'est pourquoi vous devez veiller à utiliser un formulaire sortie ne contenant que les champs ou objets saisissables que vous voulez exporter. Ne placez pas de boutons ou d'autres objets dans le formulaire sortie. Les objets de sous-formulaire sont ignorés.

L'événement Sur chargement est envoyé à la méthode du formulaire pour chaque enregistrement exporté. Utilisez cet événement pour définir les variables utilisées dans le formulaire d'export.

Le paramètre *document* peut créer un nouveau document ou désigner un document existant. Si le document a le même nom qu'un document existant, ce dernier est remplacé. Le document peut contenir un chemin d'accès de volume ou de dossiers. Si vous passez une chaîne vide, la boîte de dialogue standard de sauvegarde de fichiers est affichée. Si l'utilisateur annule ce dialogue, l'opération d'export est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'export. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Si l'export s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande **SUPPRIMER MESSAGES**.

La commande utilise par défaut le jeu de caractères UTF-8. Vous pouvez utiliser la commande **UTILISER FILTRE** pour modifier ce jeu de caractères.

Lors de l'utilisation de **EXPORTER TEXTE**, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrement est par défaut le retour chariot (code 13) sous OS X et le retour chariot+retour à la ligne (code 13 + code 10) sous Windows. Vous pouvez modifier ces valeurs en assignant de nouvelles valeurs aux deux **Variables système** *FldDelimit* et *RecDelimit*. L'utilisateur peut modifier ces valeurs par défaut dans le dialogue d'export du mode Développement. A noter que si des champs exportés contiennent des caractères définis comme délimiteurs de champ ou d'enregistrement, ces caractères sont automatiquement remplacés par des espaces dans le fichier exporté, afin de ne pas perturber le processus d'import.

Exemple

Cet exemple exporte des données vers un document texte. Cette méthode commence par le choix du formulaire sortie. Ici, vous modifiez les délimiteurs et vous effectuez l'export :

```
FORM FIXER SORTIE([Personnes];"Export")
FldDelimit:=27 ` caractère de délimitation : Escape
RecDelimit:=10 ` caractère de délimitation : Line Feed
EXPORTER TEXTE([Personnes];"Nouvelles Personnes.txt") ` Export vers le document "Nouvelles Personnes.txt"
```

Variables et ensembles système

OK prend la valeur 1 si l'export s'est correctement déroulé, sinon elle prend la valeur 0.

IMPORTER DONNEES

IMPORTER DONNEES (nomFichier {; projet {; *}})

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Chemin d'accès et nom du fichier à importer
projet	Variable texte, Variable BLOB	⇒	Contenu du projet d'import (XML ou référence d'élément DOM ou BLOB)
		⇐	Nouveau contenu du projet d'import (si le paramètre * a été passé)
*	Opérateur	⇒	Affichage de la boîte de dialogue d'import et mise à jour du projet

Description

La commande **IMPORTER DONNEES** permet d'importer des données depuis le fichier *nomFichier*. 4D peut importer des données au format Texte, Texte de longueur fixe, XML, SYLK, DIF, DBF (dBase), et 4D.

Si vous passez une chaîne vide dans le *nomFichier*, **IMPORTER DONNEES** provoque l'affichage d'une boîte de dialogue standard d'ouverture de fichiers, permettant à l'utilisateur de sélectionner le fichier d'import. Une fois la boîte de dialogue validée, la variable système Document contient le chemin d'accès et le nom du fichier d'import. Si l'utilisateur clique sur le bouton **Annuler**, l'exécution est stoppée et la variable système **OK** prend la valeur 0.

Le paramètre optionnel *projet* vous permet d'utiliser un projet pour l'import des données. Lorsque vous passez ce paramètre, l'import s'effectue directement, sans intervention de l'utilisateur (sauf si vous utilisez l'option *, cf. ci-dessous). Si vous ne passez pas ce paramètre, la boîte de dialogue de paramétrage d'import s'affiche, permettant à l'utilisateur de définir ses paramètres d'import ou de charger un projet d'import existant.

Un projet d'import contient tous les paramètres de l'import, tels que les tables et champs utilisés, les délimiteurs, etc. Vous pouvez passer dans *projet* soit une variable texte contenant du XML, soit une variable texte contenant la référence à un élément DOM préexistant, soit un BLOB. Les projets peuvent avoir été créés par programmation (projets au format XML uniquement) ou être issus du chargement de paramètres préalablement définis dans la boîte de dialogue d'import. Dans ce dernier cas, vous disposez de deux solutions :

- utiliser la commande **IMPORTER DONNEES** avec un paramètre *projet* vide et le paramètre optionnel * (cf. ci-dessous), puis stocker le paramètre *projet* résultant dans un champ Texte ou BLOB. Cette solution permet notamment de conserver le projet avec le fichier de données.
- sauvegarder le projet sur disque, puis le charger par exemple à l'aide de la commande **DOM Analyser source XML** et passer sa référence dans le paramètre *projet*.

Note de compatibilité : A compter de la version 12 de 4D, les projets d'import sont encodés en XML. 4D peut ouvrir les projets d'import générés avec des versions précédentes de 4D (format BLOB), mais les projets créés à compter de la v12 ne peuvent plus être rouverts avec une v11 ou antérieure. Il est désormais conseillé d'utiliser des variables Texte pour manipuler les fichiers d'import.

Le paramètre optionnel *, s'il est est spécifié, provoque l'affichage de la boîte de dialogue de paramétrage d'import avec les paramétrages définis dans le projet. Ce fonctionnement permet d'utiliser un projet prédéfini, tout en ayant la possibilité de modifier un ou plusieurs paramètres. En outre, le paramètre *projet* contient, après la fermeture de la boîte de dialogue d'import, les paramètres du "nouveau" projet au format XML. Vous pouvez alors le stocker dans un champ Texte, sur disque, etc.

Note : Reportez-vous à la commande **EXPORTER DONNEES** pour un exemple de définition de projet vide.

Variables et ensembles système

Si l'utilisateur clique sur **Annuler** dans une des boîtes de dialogue (de sélection de projet ou de paramétrage d'import), la variable système OK prend la valeur 0. Si l'import se déroule correctement, la variable système OK prend la valeur 1.

IMPORTER ODBC

IMPORTER ODBC (tableSource {; projet {; *}})

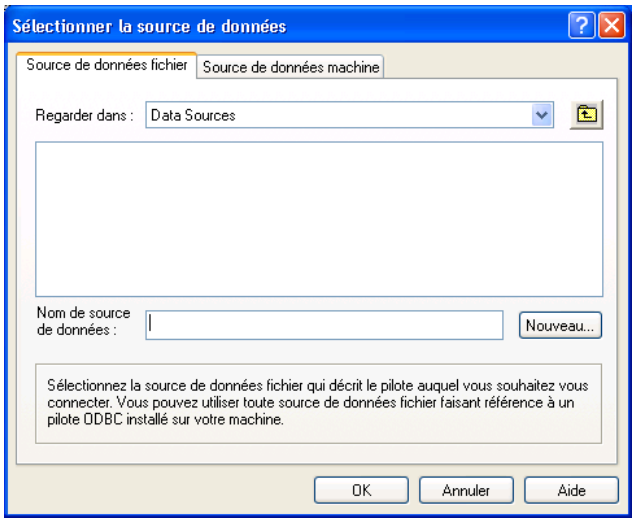
Paramètre	Type		Description
tableSource	Chaîne	⇒	Nom de la table dans la source de données ODBC
projet	BLOB	⇒	Contenu du projet d'importation
		⇐	Nouveau contenu du projet (si * est passé)
*	Opérateur	⇒	Affichage de la boîte de dialogue d'importation et mise à jour du projet

Description

La commande **IMPORTER ODBC** permet d'importer des données depuis la table *tableSource* d'une source ODBC externe.

Si vous appelez la commande **IMPORTER ODBC** en-dehors d'une connexion préalablement ouverte à l'aide de la commande **SQL LOGIN**, la boîte de dialogue de sélection de la source de données ODBC est affichée :

Windows



Mac OS

Si l'utilisateur clique sur le bouton **Annuler** dans cette boîte de dialogue, l'exécution est stoppée et la variable système OK prend la valeur 0.

Note : Cette commande ne peut pas être utilisée dans le cadre d'une connexion avec le moteur SQL interne de 4D.

Si vous ne passez pas le paramètre *projet*, 4D affiche la boîte de dialogue d'importation ODBC, permettant à l'utilisateur de configurer l'opération. Pour plus d'informations sur cette boîte de dialogue, reportez-vous au manuel Mode Développement.

Si vous passez dans le paramètre *projet* un BLOB contenant un projet d'import ODBC valide, l'importation s'effectue directement, sans intervention de l'utilisateur. Pour cela, il vous suffit de charger un projet préalablement sauvegardé sur disque dans le champ ou la variable BLOB que vous passez dans le paramètre *projet*, à l'aide de la commande **DOCUMENT VERS BLOB**. Les projets d'import ODBC sont sauvegardés depuis la boîte de dialogue d'importation ODBC.

Vous pouvez également utiliser la commande **IMPORTER ODBC** avec un paramètre *projet* vide et le paramètre optionnel *, puis stocker le paramètre projet dans un champ BLOB (cf. ci-dessous). Cette solution permet, d'une part, de le conserver avec le fichier de données et d'autre part d'éviter la phase de chargement dans un BLOB depuis le disque.

Note : Reportez-vous à la commande **EXPORTER DONNEES** pour un exemple de définition de projet vide. A noter que les projets générés dans la boîte de dialogue d'importation ODBC ne sont pas compatibles avec les commandes ou la boîte de dialogue d'importation standard de 4D.

Le paramètre optionnel *, s'il est spécifié, provoque l'affichage de la boîte de dialogue d'importation ODBC de 4D avec les paramétrages éventuellement définis dans le projet. Ce fonctionnement permet d'utiliser un projet prédéfini tout en ayant la possibilité de modifier un ou plusieurs paramètres. En outre, dans ce cas le paramètre *projet* contient, après la fermeture de la boîte de dialogue, les paramètres du "nouveau" projet. Vous pouvez alors le stocker dans un champ BLOB, dans un fichier disque, etc.

Variables et ensembles système

Si l'utilisateur clique sur le bouton **Annuler** dans une des boîtes de dialogue (de sélection de source de données ou de paramétrage d'import), la variable système OK prend la valeur 0. Si l'importation se déroule correctement, la variable système OK prend la valeur 1.

IMPORTER TEXTE

IMPORTER TEXTE ({laTable ;} nomFichier)

Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle effectuer l'import ou Table par défaut si ce paramètre est omis
nomFichier	Chaîne	⇒	Document texte à importer

Description

La commande **IMPORTER TEXTE** lit les données de *document* (document texte Windows ou Mac OS) et les écrit dans la table *laTable* en créant de nouveaux enregistrements.

L'opération d'import s'effectue par l'intermédiaire du formulaire entrée courant. Les champs et les variables sont lus en fonction du plan où ils se trouvent dans le formulaire entrée. Soyez donc attentif à ce qui se trouve au premier ou au deuxième plan en matière d'objets (champs et variables) dans le formulaire. Le premier objet dans lequel des données sont importées doit être à l'arrière-plan du formulaire. Si le nombre de champs et de variables dans le formulaire ne correspond pas au nombre de champs à importer, les champs supplémentaires sont ignorés. Un formulaire entrée utilisé pour l'import ne peut contenir de boutons. Les objets de sous-formulaire sont ignorés.

Note : Un moyen de s'assurer que les données sont importées dans les bons objets est de sélectionner l'objet dans lequel sera importé le premier champ. Cet objet doit être au premier plan. Pour l'import dans le deuxième champ, placer ce champ devant le premier et ainsi de suite, pour tous les champs et variables. Le champ qui se trouve au premier plan est donc celui dans lequel sont placées les dernières données du document à importer.

L'événement Sur validation est envoyé à la méthode du formulaire pour chaque enregistrement importé. Utilisez cet événement pour copier les données des variables vers les champs, si vous utilisez des variables dans le formulaire d'import.

Le paramètre *document* peut contenir un chemin d'accès aux noms de volumes et de dossiers. Si vous passez une chaîne vide, la boîte de dialogue standard d'ouverture de fichiers est affichée. Si l'utilisateur annule le dialogue, l'opération d'import est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'import. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Les enregistrements déjà importés le resteront. Si l'import s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande **SUPPRIMER MESSAGES**.

La commande utilise par défaut le jeu de caractères UTF-8. Vous pouvez utiliser la commande **UTILISER FILTRE** pour modifier ce jeu de caractères.

Lors de l'utilisation de **IMPORTER TEXTE**, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrements par défaut est le retour chariot (code 13). Vous pouvez modifier ces valeurs par défaut en assignant de nouvelles valeurs aux variables système **FldDelimit** et **RecDelimit**. L'utilisateur peut modifier ces valeurs par défaut dans la boîte de dialogue d'import du mode Développement. Comme les champs Texte peuvent contenir des Retours chariot, soyez prudent si vous utilisez le Retour chariot comme délimiteur entre les champs à importer.

Exemple

Cet exemple importe des données d'un document texte. Cette méthode commence par le choix du formulaire. Ici, vous changez les délimiteurs, et vous faites l'import :

```
FORM FIXER ENTREE([Personnes];"Import")
FldDelimit:=27 ` caractère de délimitation : Escape
RecDelimit:=10 ` caractère de délimitation : Line Feed
IMPORTER TEXTE([Personnes];"Nouvelles Personnes.txt") ` Import du document "Nouvelles Personnes.txt"
```

Variables et ensembles système

OK prend la valeur 1 si l'import s'est correctement déroulé, sinon elle prend la valeur 0.

LECTURE DIF

LECTURE DIF ({laTable ;} nomFichier)

Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle effectuer l'import ou Table par défaut si ce paramètre est omis
nomFichier	Chaîne	⇒	Document DIF à importer

Description

La commande **LECTURE DIF** lit les données de *document* (document DIF Windows ou Mac OS) et les écrit dans la table *laTable* en créant de nouveaux enregistrements.

L'opération d'import s'effectue par l'intermédiaire du formulaire entrée courant. Les champs et les variables sont lus en fonction du plan où ils se trouvent dans le formulaire entrée. Soyez donc attentif à ce qui se trouve au premier ou au deuxième plan en matière d'objets (champs et variables) dans le formulaire. Le premier objet dans lequel des données sont importées doit être à l'arrière-plan du formulaire. Si le nombre de champs et de variables dans le formulaire ne correspond pas au nombre de champs à importer, les champs supplémentaires sont ignorés. Un formulaire entrée utilisé pour l'import ne peut contenir de boutons. Les objets de sous-formulaire sont ignorés.

Note : Un moyen de s'assurer que les données sont importées dans les bons objets est de sélectionner l'objet dans lequel sera importé le premier champ. Cet objet doit être au premier plan. Pour l'import dans le deuxième champ, placer ce champ devant le premier et ainsi de suite, pour tous les champs et variables. Le champ qui se trouve au premier plan est donc celui dans lequel sont placées les dernières données du document à importer.

L'événement Sur validation est envoyé à la méthode du formulaire pour chaque enregistrement importé. Utilisez cet événement pour copier les données des variables vers les champs, si vous utilisez des variables dans le formulaire d'import.

Le paramètre *document* peut contenir un chemin d'accès aux noms de volumes et de dossiers. Si vous passez une chaine vide, la boîte de dialogue standard d'ouverture de fichiers est affichée. Si l'utilisateur annule le dialogue, l'opération d'import est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'import. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Les enregistrements déjà importés le resteront. Si l'import s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande **SUPPRIMER MESSAGES**.

La commande utilise par défaut le jeu de caractères UTF-8. Les documents au format DIF utilisant généralement le jeu de caractères IBM437, il peut être nécessaire d'utiliser la commande **UTILISER FILTRE** pour définir le jeu de caractères approprié.

Lors de l'utilisation de **LECTURE DIF**, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrements par défaut est le retour chariot (code 13). Vous pouvez modifier ces valeurs par défaut en assignant de nouvelles valeurs aux variables système **FldDelimit** et **RecDelimit**. L'utilisateur peut modifier ces valeurs par défaut dans la boîte de dialogue d'import du mode Développement. Comme les champs Texte peuvent contenir des Retours chariot, soyez prudent si vous utilisez le Retour chariot comme délimiteur entre les champs à importer.

Exemple

Cet exemple importe des données d'un document DIF. Cette méthode commence par le choix du formulaire, puis effectue l'import :

```
FORM FIXER ENTREE([Personnes];"Import")
LECTURE DIF([Personnes];"Nouvelles Personnes.dif") ` Import du document "Nouvelles Personnes.dif"
```

Variables et ensembles système

OK prend la valeur 1 si l'import s'est correctement déroulé, sinon elle prend la valeur 0.

LECTURE SYLK

LECTURE SYLK ({laTable ;} nomFichier)

Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle effectuer l'import ou Table par défaut si ce paramètre est omis
nomFichier	Chaîne	⇒	Document SYLK à importer

Description

La commande **LECTURE SYLK** lit les données de *document* (document SYLK Windows ou Mac OS) et les écrit dans la table *laTable* en créant de nouveaux enregistrements.

L'opération d'import s'effectue par l'intermédiaire du formulaire entrée courant. Les champs et les variables sont lus en fonction du plan où ils se trouvent dans le formulaire entrée. Soyez donc attentif à ce qui se trouve au premier ou au deuxième plan en matière d'objets (champs et variables) dans le formulaire. Le premier objet dans lequel des données sont importées doit être à l'arrière-plan du formulaire. Si le nombre de champs et de variables dans le formulaire ne correspond pas au nombre de champs à importer, les champs supplémentaires sont ignorés. Un formulaire entrée utilisé pour l'import ne peut contenir de boutons. Les objets de sous-formulaire sont ignorés.

Note : Un moyen de s'assurer que les données sont importées dans les bons objets est de sélectionner l'objet dans lequel sera importé le premier champ. Cet objet doit être au premier plan. Pour l'import dans le deuxième champ, placer ce champ devant le premier et ainsi de suite, pour tous les champs et variables. Le champ qui se trouve au premier plan est donc celui dans lequel sont placées les dernières données du document à importer.

L'événement Sur validation est envoyé à la méthode du formulaire pour chaque enregistrement importé. Utilisez cet événement pour copier les données des variables vers les champs, si vous utilisez des variables dans le formulaire d'import.

Le paramètre *document* peut contenir un chemin d'accès aux noms de volumes et de dossiers. Si vous passez une chaine vide, la boîte de dialogue standard d'ouverture de fichiers est affichée. Si l'utilisateur annule le dialogue, l'opération d'import est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'import. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Les enregistrements déjà importés le resteront. Si l'import s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande **SUPPRIMER MESSAGES**.

La commande utilise par défaut le jeu de caractères UTF-8. Les documents au format SYLK utilisant généralement le jeu de caractères ISO-8859-1, il peut être nécessaire d'utiliser la commande **UTILISER FILTRE** pour définir le jeu de caractères approprié.

Lors de l'utilisation de **LECTURE SYLK**, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrements par défaut est le retour chariot (code 13). Vous pouvez modifier ces valeurs par défaut en assignant de nouvelles valeurs aux variables système **FldDelimit** et **RecDelimit**. L'utilisateur peut modifier ces valeurs par défaut dans la boîte de dialogue d'import du mode Développement. Comme les champs Texte peuvent contenir des Retours chariot, soyez prudent si vous utilisez le Retour chariot comme délimiteur entre les champs à importer.

Exemple

Cet exemple importe des données d'un document SYLK. Cette méthode commence par le choix du formulaire puis déclenche l'import :

```
FORM FIXER ENTREE([Personnes];"Import")
LECTURE SYLK([Personnes];"Nouvelles Personnes.slk") ` Import du document "Nouvelles Personnes.slk"
```

Variables et ensembles système

OK prend la valeur 1 si l'import s'est correctement déroulé, sinon elle prend la valeur 0.

Impressions

- Intégration du pilote PDFCreator sous Windows
- BLOB vers paramètres impression Nouveauté 16.0
- CUMULER SUR
- Est en apercu impression
- FERMER TACHE IMPRESSION
- FIXER APERCU IMPRESSION
- FIXER IMPRIMANTE COURANTE
- FIXER MARGE IMPRESSION
- FIXER OPTION IMPRESSION Modifié 16.0
- FIXER TAQUET IMPRESSION
- IMPRIMER ENREGISTREMENT
- IMPRIMER ETIQUETTES
- Imprimer ligne
- Imprimer objet
- IMPRIMER SELECTION
- Lire apercu impression
- Lire hauteur imprimee
- Lire imprimante courante
- LIRE MARGE IMPRESSION
- LIRE OPTION IMPRESSION Modifié 16.0
- Lire taquet impression
- LIRE ZONE IMPRESSION
- LISTE IMPRIMANTES Modifié 16.0
- Niveau
- NIVEAUX DE RUPTURES
- OUVRIR TACHE IMPRESSION
- Page impression
- PARAMETRES IMPRESSION
- Paramètres impression vers BLOB Nouveauté 16.0
- SAUT DE PAGE
- Sous total
- UTILISER PARAMETRES IMPRESSION
- VALEURS OPTION IMPRESSION

Intégration du pilote PDFCreator sous Windows

La prise en charge des impressions PDF diffère selon la version de Windows :

- jusqu'à Windows 8 inclus, il est nécessaire d'utiliser le pilote PDFCreator.
- à compter de Windows 10, un pilote natif Microsoft est intégré.

Note : Sous Mac OS, l'impression PDF est prise en charge de façon native par le système.

Windows 8 et versions précédentes

La prise en charge des impressions PDF sous Windows s'appuie sur le pilote (*driver*) PDFCreator afin de proposer des fonctions d'impression PDF simples et fonctionnelles. Les commandes **LIRE OPTION IMPRESSION** et **FIXER OPTION IMPRESSION** tirent directement parti de ce pilote.

PDFCreator est un pilote gratuit (OpenSource) régi par la licence AFPL (Aladdin Free Public License). Pour pouvoir utiliser le pilote PDFCreator, vous devez le télécharger et l'installer dans votre environnement, il n'est pas installé par défaut par 4D. Vous devez disposer d'un accès Administrateur pour pouvoir l'installer. Vous pouvez télécharger PDFCreator à l'adresse http://sourceforge.net/projects/pdfcreator/files/PDFCreator/

Attention, vous devez utiliser une version de PDFCreator **compatible avec 4D**. Pour connaître les versions compatibles et certifiées de PDFCreator, veuillez consulter les matrices de certification des produits 4D, disponibles dans la **page Ressources** (**rubrique Compatibilité**) du site Web de 4D.

Au cours de l'installation, une nouvelle imprimante virtuelle nommée par défaut "PDFCreator" est installée dans votre système. Vous pouvez changer ce nom si vous le souhaitez.

A partir de Windows 10

Windows 10 inclut un pilote d'impression PDF natif, permettant à 4D de créer directement des PDFs sans qu'il soit nécessaire d'utiliser un pilote tiers comme PDFCreator.

Le nom du pilote est "Microsoft Print to PDF".

Voici un exemple de création d'un document PDF sous Windows 10 via les commandes d'impression de 4D :

```
$pdfpath:=Dossier systeme(Bureau)+"test.pdf"

$pdfprintername:="Microsoft Print to PDF"
TABLEAU TEXTE($name1;0)
LISTE IMPRIMANTES($name1)
Si(Chercher dans tableau($name1;$pdfprintername)>0)
   FIXER IMPRIMANTE COURANTE($pdfprintername)
   FIXER OPTION IMPRESSION(Option destination;2;$pdfpath)
   TOUT SELECTIONNER([Table_1])
   IMPRIMER SELECTION([Table_1];*)
   FIXER IMPRIMANTE COURANTE("")
Fin de si
```

BLOB vers paramètres impression

BLOB vers paramètres impression (paramImpression {; param}) -> Résultat

Paramètre	Type		Description
paramImpression	BLOB	⇒	BLOB contenant les paramètres d'impression
param	Entier long	⇒	0=Utilise les valeurs sauvegardées pour le nombre de copies et la plage d'impression, 1=Réinitialise aux valeurs par défaut
Résultat	Entier long	⇐	Code d'état : 1=Opération réussie, 0=Pas d'imprimante courante, -1=Paramètres incorrects, 2=L'imprimante a changé

Description

La commande **BLOB vers paramètres impression** remplace les paramètres d'impression courants de 4D par les paramètres stockés dans le BLOB *paramImpression*. Ce BLOB doit avoir été généré par la commande **Paramètres impression vers BLOB** ou par la commande de 4D Pack **_o_AP Print settings to BLOB** (voir ci-dessous).

Le paramètre *param* permet de définir la façon de gérer les paramètres de base "nombre de copies" et "plage d'impression" :

- si vous passez 0 ou omettez ce paramètre, les valeurs stockées dans le BLOB sont utilisées pour l'impression.
- si vous passez 1, les valeurs sont réinitialisées aux valeurs par défaut : le nombre de copies est fixé à 1, et la plage d'impression est fixée à "toutes les pages".

Les paramètres d'impression s'appliquent à l 'imprimante courante et durant toute la session, jusqu'à ce qu'une commande telle que **UTILISER PARAMETRES IMPRESSION**, **FIXER OPTION IMPRESSION** ou **IMPRIMER SELECTION** sans le paramètre > les modifie. Les paramètres fixés sont utilisés plus particulièrement par les commandes **IMPRIMER SELECTION**, **IMPRIMER ETIQUETTES**, **IMPRIMER ENREGISTREMENT**, **Imprimer ligne** et **QR ETAT**, ainsi que par les commandes d'impression dans les menus de 4D, y compris ceux de l'environnement Développement.

Les commandes **IMPRIMER SELECTION**, **IMPRIMER ETIQUETTES** et **IMPRIMER ENREGISTREMENT** doivent être appelées avec le paramètre > (si applicable) de façon à ce que les paramètres définis par **BLOB vers paramètres impression** soient gardés.

La commande retourne un des codes d'état suivants :

- -1 : le BLOB est incorrect.
- 0 : aucune imprimante courante n'est sélectionnée (dans ce cas, la commande ne fait rien).
- 1 : le BLOB a été correctement chargé.
- 2 : le BLOB a été correctement chargé mais le nom de l'imprimante courante a changé (*).
  Note : le code (2) est toujours retourné si le BLOB a été crée avec la commande de 4D Pack **_o_AP Print settings to BLOB**, même si le nom de l'imprimante n'a pas changé, car cette information n'est pas présente dans les BLOBs de 4D Pack.

(*) Les paramètres dépendent de l'imprimante courante sélectionnée au moment où le BLOB a été stocké. Appliquer ces paramètres à une autre imprimante sera pris en charge si les deux imprimantes sont du même modèle. Si les imprimantes sont différentes, seuls les paramètres communs seront restaurés.

Windows / OS X

Le BLOB *paramImpression* peut être sauvegardé et lu sur les deux plate-formes. Toutefois, même si certains paramètres d'impression sont communs, d'autres sont spécifiques à la plate-forme et dépendent du pilote d'impression et des versions de l'OS. Si le même BLOB *paramImpression* est partagé entre les deux plate-formes, vous pouvez perdre des informations.*

Lorsque vous utilisez un environnement hétérogène, pour restaurer le maximum de paramètres d'impression disponibles pour chaque plate-forme (et pas seulement la partie commune), il est recommandé de gérer deux BLOBs *paramImpression*, un pour chaque plate-forme.

Compatibilité avec les commandes 4D Pack

Les BLOBs de paramètres d'impression générés avec la commande 4D Pack **_o_AP Print settings to BLOB** peuvent être chargés et utilisés par la commande **BLOB vers paramètres impression**. Notez toutefois que s'ils sont stockés avec **Paramètres impression vers BLOB**, ils sont convertis et ne pourront plus être ouvert avec **_o_AP BLOB to print settings**. La commande **BLOB vers paramètres impression** stocke davantage d'informations que la commande **_o_AP Print settings to BLOB**.

Exemple

Vous voulez appliquer des paramètres d'impression précédemment stockés sur disque au dialogue d'impression de 4D :

```
C_BLOB(curSettings)
DOCUMENT VERS BLOB(Dossier 4D(Dossier 4D actif)+"current4Dsettings.blob";curSettings)
 //current4Dsettings a été créé avec la commande Paramètres impression vers BLOB
$err:=BLOB vers paramètres impression(curSettings;0)
Au cas ou
   :($err=1)
 //tout est OK
   :($err=2)
      CONFIRMER("L'imprimante a changé. \n\nVérifiez les paramètres d'impression.")
      Si(OK=1)
         PARAMETRES IMPRESSION
      Fin de si
   :($err=0)
      ALERTE("Il n'y a pas d'imprimante courante.")
   :($err=-1)
      ALERTE("Fichier de paramètres invalide.")
Fin de cas
```

CUMULER SUR

CUMULER SUR (objet {; objet2 ; ... ; objetN})

Paramètre	Type		Description
objet	Champ, Variable	⇨	Champ ou variable de type numérique à cumuler

Description

CUMULER SUR désigne les champ(s) ou variable(s) à cumuler dans un état créé à l'aide de la commande **IMPRIMER SELECTION**.

Vous **devez** appeler **NIVEAUX DE RUPTURES** et **CUMULER SUR** avant la génération de chaque état dans lequel vous voulez utiliser des ruptures. Ces commandes activent le traitement des ruptures pour un état. Pour plus d'informations, reportez-vous à la description de la commande **Sous total**.

Utilisez **CUMULER SUR** si vous souhaitez calculer des sous-totaux pour des champs ou variables numériques dans un état. **CUMULER SUR** indique à 4D qu'il faut stocker les sous-totaux pour chaque élément spécifié dans *objet*. Les sous-totaux sont cumulés pour chaque niveau de rupture spécifié par la commande **Sous total**.

Exécutez **CUMULER SUR** avant d'imprimer un état à l'aide de **IMPRIMER SELECTION**.

Utilisez la fonction **Sous total** dans la méthode formulaire ou une méthode objet pour retourner le sous-total d'un des objets spécifié dans *objet*.

Exemple

Reportez-vous à l'exemple de la commande **NIVEAUX DE RUPTURES**.

Est en apercu impression

Est en apercu impression -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Vrai = Impression à l'écran, Faux = Pas d'impression écran

Description

La commande **Est en apercu impression** retourne Vrai si l'option **Aperçu avant impression** est cochée dans la boîte de dialogue d'impression, et Faux sinon. Ce paramétrage est local au process.

A la différence de la commande **Lire apercu impression**, **Est en apercu impression** retourne la valeur finale de l'option, après validation de la boîte de dialogue par l'utilisateur. Cette commande vous permet donc de déterminer avec certitude si l'impression a effectivement lieu en mode "aperçu".

Exemple

Cet exemple permet de prendre en compte tous les types d'impressions :

```
FIXER APERCU IMPRESSION(Vrai) //Impression écran par défaut
PARAMETRES IMPRESSION
Si(OK=1)
 //L'utilisateur peut avoir changé la destination d'impression
   Si(Est en apercu impression)  // Vrai si aperçu
      FORM FIXER SORTIE([Factures];"versEcran")
   Sinon
      FORM FIXER SORTIE([Factures];"versImprimante")
   Fin de si
   OUVRIR TACHE IMPRESSION
   TOUT SELECTIONNER([Factures])
   IMPRIMER SELECTION([Factures];>)
   FERMER TACHE IMPRESSION
Fin de si
```

FERMER TACHE IMPRESSION

FERMER TACHE IMPRESSION
Ne requiert pas de paramètre

Description

La commande **FERMER TACHE IMPRESSION** permet de refermer la tâche d'impression préalablement ouverte par la commande **OUVRIR TACHE IMPRESSION** et d'envoyer à l'imprimante courante le document d'impression éventuellement construit.

Une fois cette commande exécutée, l'imprimante redevient disponible pour d'autres impressions.

FIXER APERCU IMPRESSION

FIXER APERCU IMPRESSION (aperçu)

Paramètre	Type		Description
aperçu	Booléen	⇨	Impression à l'écran (Vrai) ou non (Faux)

Description

La commande **FIXER APERCU IMPRESSION** vous permet de sélectionner ou de désélectionner l'option d'**aperçu** dans la boîte de dialogue standard d'impression. Si vous passez Vrai dans *aperçu*, l'option "à l'écran" sera cochée. Si vous passez Faux, elle ne sera pas cochée. Ce paramétrage est local au process et n'affecte pas les autres process ou utilisateurs.

Exemple

L'exemple suivant sélectionne l'option **A l'écran** pour afficher le résultat d'une recherche de clients à l'écran, puis la désélectionne.

```
CHERCHER([Clients])
Si(OK=1)
   FIXER APERCU IMPRESSION(Vrai)
   IMPRIMER SELECTION([Clients];*)
   FIXER APERCU IMPRESSION(Faux)
Fin de si
```

FIXER IMPRIMANTE COURANTE

FIXER IMPRIMANTE COURANTE (nomImpr)

Paramètre	Type		Description
nomImpr	Chaîne	➡	Nom de l'imprimante à utiliser

Description

La commande **FIXER IMPRIMANTE COURANTE** permet de désigner l'imprimante à utiliser pour les impressions avec l'application 4D courante.

Passez dans le paramètre *nomImpr* le nom de l'imprimante à sélectionner. Pour obtenir la liste des imprimantes disponibles, utilisez au préalable la commande **LISTE IMPRIMANTES**.
Si vous passez une chaîne vide dans *nomImpr*, l'imprimante courante définie dans le système sera utilisée.
FIXER IMPRIMANTE COURANTE vous permet de désigner l'imprimante PDF générique du système afin d'effectuer des impressions PDF. La valeur à utiliser dépend de la version de l'OS et de celle de 4D.

- **Windows 8 et versions précédentes :**
  4D s'appuie sur le pilote PDFCreator pour l'impression de documents PDF sous Windows (cf. section **Intégration du pilote PDFCreator sous Windows**). Pour imprimer un document PDF, passez dans le paramètre *nomImpr* le nom de l'imprimante virtuelle installée par PDFCreator. Par défaut, le nom de l'imprimante virtuelle est "PDFCreator". Toutefois, ce nom peut avoir été modifié au moment de l'installation du pilote. Pour que 4D recherche et utilise automatiquement le nom de l'imprimante virtuelle, même s'il a été personnalisé, vous pouvez passer dans *nomImpr* la constante suivante (thème **Options d'impression**) :

Constante	Type	Valeur
PDFCreator Nom imprimante	Chaîne	PDFCreator

- **A partir de Windows 10 :**
  Windows 10 inclut un pilote d'impression PDF natif, permettant à 4D de créer directement des PDFs sans qu'il soit nécessaire d'utiliser un pilote tiers comme PDFCreator.
  Le nom du pilote est "Microsoft Print to PDF" (cf. exemple présenté dans la section **Intégration du pilote PDFCreator sous Windows**).

- **Sous OS X et à partir de Windows 10 (4D Developer Edition v15 R5 64 bits et suivantes) :**
  Une constante du thème **Options d'impression** vous permet de désigner automatiquement l'imprimante PDF générique, quelle que soit la plate-forme. Ce principe facilite l'écriture de code générique.

Constante	Type	Valeur	Comment
Driver PDF générique	Chaîne	_4d_pdf_printer	**Note :** Cette fonctionnalité n'est pas disponible dans les versions 32 bits de 4D. ◦ Sous OS X, déclare le pilote par défaut comme imprimante courante. Ce pilote n'est pas visible et ne se trouve pas dans la liste retournée par la commande **LISTE IMPRIMANTES**. Notez que le chemin d'accès pour le document PDF doit être défini en utilisant la commande **FIXER OPTION IMPRESSION**, sinon l'erreur 3107 est retournée. ◦ Sous Windows, déclare le pilote PDF de Windows (nommé "Microsoft Print to PDF") comme imprimante courante. Cette constante est disponible sous Windows 10 uniquement, lorsque l'option PDF est installée. Avec d'autres versions de Windows, ou lorsqu'il n'y a pas de pilote PDF disponible, la commande ne fait rien et la variable système *OK* prend la valeur 0.

La commande **FIXER IMPRIMANTE COURANTE** doit être appelée avant **FIXER OPTION IMPRESSION** afin que les options disponibles correspondent à l'imprimante sélectionnée. En revanche, **FIXER IMPRIMANTE COURANTE** doit être appelée après **UTILISER PARAMETRES IMPRESSION** (le cas échéant), sinon le paramétrage d'imprimante n'est pas conservé.

Cette commande peut être utilisée avec les commandes **IMPRIMER SELECTION**, **IMPRIMER ENREGISTREMENT**, **Imprimer ligne**, **QR ETAT** et s'applique à toutes les impressions de 4D, y compris en mode Développement.
Les commandes d'impression doivent impérativement être appelées avec le paramètre > (le cas échéant) afin que le paramétrage défini soit conservé.

Variables et ensembles système

Si la sélection d'imprimante est correctement effectuée, la variable système OK prend la valeur 1. Dans le cas contraire (par exemple l'imprimante désignée est introuvable), la variable système OK prend la valeur 0 et l'imprimante courante est inchangée.

Exemple

Création d'un document PDF sous Windows 10 avec 4D Developer Edition 64 bits :

```
C_TEXTE($pdfpath)
$pdfpath:=Dossier systeme(Bureau)+"test.pdf"
FIXER IMPRIMANTE COURANTE(Driver PDF générique)
FIXER OPTION IMPRESSION(Option destination;2;$pdfpath)
TOUT SELECTIONNER([Table_1])
IMPRIMER SELECTION([Table_1];*)
FIXER IMPRIMANTE COURANTE("")
```

FIXER MARGE IMPRESSION

FIXER MARGE IMPRESSION (gauche ; haut ; droit ; bas)

Paramètre	Type		Description
gauche	Entier long	➡	Marge gauche
haut	Entier long	➡	Marge supérieure
droit	Entier long	➡	Marge droite
bas	Entier long	➡	Marge inférieure

Description

La commande **FIXER MARGE IMPRESSION** permet de fixer les valeurs des différentes marges d'impression lors de l'utilisation des commandes **Imprimer ligne**, **IMPRIMER SELECTION** et **IMPRIMER ENREGISTREMENT**.

Vous pouvez passer dans les paramètres *gauche*, *haut*, *droite* et *bas* une des valeurs suivantes :

- 0 = utiliser les marges papier
- -1 = utiliser les marges imprimante
- valeur > 0 = marge en pixels (rappelons qu'1 pixel en 72 dpi représente approximativement 0,4 mm)

Les valeurs des paramètres *droite* et *bas* sont relatives à la droite et au bas du papier.

Par défaut, 4D base ses impressions sur les marges imprimante. Une fois la commande **FIXER MARGE IMPRESSION** exécutée, les paramètres modifiés seront conservés dans le même process pour toute la session.

Note : Pour plus d'informations sur la gestion des marges d'impression dans 4D, reportez-vous à la description de la commande **LIRE MARGE IMPRESSION**.

Exemple 1

L'exemple suivant permet d'obtenir la taille de la marge morte :

```
FIXER MARGE IMPRESSION(-1;-1;-1;-1) `Fixe la marge imprimante
LIRE MARGE IMPRESSION($g;$h;$d;$b)
 `$g, $h, $d et $b correspondent aux marges mortes de la feuille
```

Exemple 2

L'exemple suivant permet d'obtenir la taille du papier :

```
FIXER MARGE IMPRESSION(0;0;0;0) `Fixe la marge papier
LIRE ZONE IMPRESSION($hauteur;$largeur)
 `Pour du A4 : $hauteur=842 ; $largeur=595 pixels
```

FIXER OPTION IMPRESSION

FIXER OPTION IMPRESSION (option ; valeur1 {; valeur2})

Paramètre	Type		Description
option	Entier long	⇒	Numéro d'option ou Code d'option PDF
valeur1	Entier long, Texte	⇒	Valeur 1 de l'option
valeur2	Entier long, Texte	⇒	Valeur 2 de l'option

Description

La commande **FIXER OPTION IMPRESSION** permet de modifier par programmation la valeur d'une option d'impression. Chaque option définie à l'aide de cette commande est appliquée dans toute la base et durant toute la session tant qu'aucune autre commande modifiant les paramètres d'impression (**PARAMETRES IMPRESSION**, **IMPRIMER SELECTION** sans le paramètre >, etc.) n'est appelée. Si une tâche d'impression a été ouverte, l'option est définie pour la tâche et n'est pas modifiable tant que la tâche n'est pas terminée.

Le paramètre *option* vous permet de désigner l'option à modifier. Vous pouvez passer soit une des constantes prédéfinies du thème **Options d'impression**, soit un code d'option PDF (utilisable avec le pilote PDFCreator sous Windows uniquement).
Passez dans les paramètres *valeur1* et (facultativement) *valeur2* la ou les nouvelle(s) valeur(s) de l'*option* spécifiée. Le nombre et la nature des valeurs à passer dépend du type d'option spécifiée.

Utiliser un numéro d'option (constante)

Le tableau suivant liste les options et leurs valeurs possibles :

Constante	Type	Valeur	Comment
Option papier	Entier long	1	Si vous passez uniquement *valeur1*, il contient le nom du papier. Si vous passez les deux paramètres, *valeur1* contient la largeur du papier et *valeur1* contient la hauteur du papier. La largeur et la hauteur sont exprimées en pixels écran. Utilisez la commande **VALEURS OPTION IMPRESSION** pour connaître le nom, la hauteur et la largeur de tous les formats de papier proposés par l'imprimante.
Option orientation	Entier long	2	*valeur1* uniquement : 1=Portrait, 2=Paysage. Si une option d'orientation différente est utilisée, **LIRE OPTION IMPRESSION** retourne 0 dans *valeur1*. **Versions 64 bits** : Cette option peut être appelée au sein d'une tâche d'impression, ce qui signifie que vous pouvez passer du mode portrait au mode paysage et inversement dans la même tâche d'impression.
Option échelle	Entier long	3	*valeur1* uniquement : valeur d'échelle en pourcentage. Attention, certaines imprimantes ne permettent pas de modifier l'échelle. Si vous passez une valeur invalide, la propriété est remise à 100% au moment de l'impression.
Option nombre copies	Entier long	4	*valeur1* uniquement : nombre de copies à imprimer
Option alimentation	Entier long	5	(Windows uniqument) *valeur1* uniquement : numéro correspondant à l'indice, dans le tableau des bacs retourné par la commande **VALEURS OPTION IMPRESSION**, du bac papier à utiliser. Cette option est utilisable sous Windows uniquement.
Option couleur	Entier long	8	(Windows uniquement) *valeur1* uniquement : code indiquant le mode de prise en charge de la couleur : 1=Noir et blanc (monochrome), 2=Couleur. **Versions 64 bits** : Cette option n'est pas prise en charge dans les versions 64 bits de 4D (obsolète).
Option destination	Entier long	9	*valeur1*: code indiquant le type de destination de l'impression : 1=Imprimante, 2=Fichier (PC)/PS (Mac), 3=Fichier PDF, 5=Ecran (option du pilote OS X) Si *valeur1* est différent de 1 ou de 5, *valeur2* contient un chemin d'accès pour le document résultant. Ce chemin sera utilisé jusqu'à ce qu'un autre chemin soit spécifié. Si un fichier du même nom existe déjà à l'emplacement de destination, il est remplacé. Avec **LIRE OPTION IMPRESSION**, si la valeur courante n'est pas dans la liste prédéfinie, *valeur1* contient -1 et la variable système OK vaut 1. Si une erreur se produit, *valeur1* et la variable système OK valent 0. **Note :** Sous Windows, vous pouvez définir la destination d'impression 3 (Fichier PDF) lorsque le pilote PDF Creator a été installé. Lorsque les valeurs (*9;3;chemin*) sont passées, 4D lance automatiquement une impression PDF "silencieuse" et prend en compte les codes d'options éventuellement passés (à noter que si vous passez une chaîne vide dans *valeur2* ou omettez ce paramètre, une boîte de dialogue d'enregistrement de fichier apparaît au moment de l'impression). A l'issue de l'impression, les paramétrages courants sont restaurés. Ce principe simplifie le pilotage des impressions PDF par 4D et permet d'écrire du code multi-plate-forme. Si les valeurs (*9;3;chemin*) ne sont pas passées, l'impression n'est pas pilotée par 4D et les éventuels codes d'options de PDF Creator sont ignorés.
Option recto verso	Entier long	11	(Windows uniquement) *valeur1*: 0=Recto ou standard, 1=Recto-verso. Si *valeur1*=1, *valeur2* contient la reliure à appliquer : 0=Reliure à gauche (valeur par défaut), 1=Reliure en haut. **Note :** Cette option est utilisable sous Windows uniquement.
Option nom document à imprimer	Entier long	12	*valeur1* uniquement : nom du document d'impression, qui apparaît dans la liste des documents du serveur d'impression. Le nom défini par cette instruction sera utilisé pour tous les documents d'impression de la session tant qu'un nouveau nom ou une chaîne vide ne sera pas passé(e). Pour utiliser ou rétablir le fonctionnement standard (utilisation du nom de la méthode dans le cas d'une méthode, nom de la table pour un enregistrement, etc.), passez une chaîne vide dans *valeur1*.
Option mode impression Mac	Entier long	13	(Mac uniquement) *valeur1* uniquement : 0=impression en mode PDF (valeur par défaut), 1=impression en mode Postscript. **Notes :** - Cette option n'a pas d'effet sous Windows. - Sous OS X, les impressions sont effectuées par défaut en mode PDF. Or, le pilote d'impression PDF ne prend pas en charge les images PICT encapsulant des informations Postscript — ces images sont générées notamment par des logiciels de dessin vectoriel. Pour résoudre ce problème, cette option permet de modifier le mode d'impression sous OS X pour la session courante. Attention, l'impression en mode Postscript peut entraîner des effets de bords indésirables. **Versions 64 bits :** Cette option n'est pas prise en charge ; elle est remplacée par l'utilisation de l'option Driver PDF générique de la commande **FIXER IMPRIMANTE COURANTE**.
Option masquer progression impr	Entier long	14	*valeur1* uniquement : 1=masquer toutes les fenêtres de progression d'impression, 0=afficher les fenêtres de progression d'impression (fonctionnement par défaut). Cette option est particulièrement utile dans le cadre des impressions en PDF sous OS X. **Note :** Il existe déjà une option d'affichage Progression de l'impression accessible via la boîte de dialogue des Propriétés de la base (page Interface). Toutefois, elle est globale à l'application et ne masque pas toutes les fenêtres sous OS X.
Option intervalle de page	Entier long	15	*valeur1*=numéro de la première page à imprimer (valeur par défaut 1) et (optionnel) *valeur2*=numéro de la dernière page à imprimer (valeur par défaut -1 = fin du document).
Option ancienne couche impression	Entier long	16	(Versions 4D 64 bits pour Windows uniquement) *valeur1* uniquement : 1=sélectionner l'ancienne couche d'impression GDI pour toutes les tâches d'impression suivantes, 0=sélectionner la couche d'impression D2D (défaut). **Versions 64 bits :** Ce sélecteur est pris en charge dans les applications 4D 64 bits monopostes sous Windows uniquement, et est ignoré pour les autres plates-formes. Il est principalement destiné, dans ces applications, à permettre aux plug-ins d'ancienne génération d'imprimer dans des tâches d'impression 4D.

Une fois fixée à l'aide de cette commande, une option d'impression sera conservée durant toute la session pour l'application 4D entière. Elle sera utilisée par les commandes **IMPRIMER SELECTION**, **IMPRIMER ENREGISTREMENT**, **Imprimer ligne**, **QR ETAT** et par toutes les impressions de 4D, y compris en mode Développement.

Notes :

- Il est impératif d'utiliser le paramètre optionnel > avec les commandes **IMPRIMER SELECTION**, **IMPRIMER ENREGISTREMENT** et **SAUT DE PAGE** afin de ne pas réinitialiser les options d'impression définies à l'aide de la commande **FIXER OPTION IMPRESSION**.
- La commande **FIXER OPTION IMPRESSION** prend principalement en charge les imprimantes PostScript. Elle peut être utilisée avec d'autres types d'imprimantes, telles que PCL ou Ink, mais dans ce cas il est possible que certaines options ne soient pas disponibles.

Utiliser un code d'option PDF (Windows)

Pour pouvoir utiliser un code d'option PDF dans le paramètre *option*, vous devez avoir installé le pilote PDFCreator dans votre environnement 4D (pour plus d'informations, reportez-vous à la section **Intégration du pilote PDFCreator sous Windows**). De plus, pour que le code d'option soit pris en compte, vous devez avoir activé le pilotage de l'impression PDF par 4D via l'instruction suivante :

```
FIXER OPTION IMPRESSION(Option destination;3;nomFichier)
```

Dans le cas contraire, les codes d'options sont ignorés.

Un code d'option PDF est une valeur de type texte constituée de deux parties, *TypeOption* et *NomOption*, assemblées sous la forme "*TypeOption:NomOption*". Voici la description de ce code :

- *TypeOption* indique si vous désignez une option native de PDFCreator ou une option d'administration PDF de 4D. Deux valeurs sont acceptées :
  - **PDFOptions** = option native
  - **PDFInfo** = option interne.
- *NomOption* désigne l'option à définir (dépend de la valeur de *TypeOption*).
  - Si *TypeOption* = **PDFOptions**, vous pouvez passer dans *NomOption* l'une des nombreuses options natives de PDFCreator. Par exemple, l'option UseAutosave agit sur la sauvegarde automatique. Pour pouvoir modifier cette option, passez "PDFOptions:UseAutosave" dans le paramètre *option* et la valeur à utiliser dans le paramètre *valeur1*. Pour une description complète des options natives de PDFCreator, reportez-vous à la documentation livrée avec le pilote PDFCreator.
  - Si *TypeOption* = **PDFInfo**, vous pouvez passer dans *NomOption* un des sélecteurs spécifiques suivants :
    - **Reset print** : permet de réinitialiser le statut d'attente interne afin, notamment, de sortir d'une boucle infinie. Dans ce cas, *valeur1* n'est pas utilisé.
    - **Reset standard options** : permet de rétablir toutes les options de PDFCreator à leurs valeurs par défaut. Si une impression est en cours, les paramètres par défaut sont appliqués à l'issue de l'impression. Dans ce cas, *valeur1* n'est pas utilisé.
    - **Start** : permet de démarrer ou de stopper le spouler de PDFCreator. Passez 0 dans *valeur1* pour le stopper et 1 pour le démarrer.
    - **Reset options** : permet de réinitialiser toutes les options modifiées depuis le début de la session à l'aide de la commande **FIXER OPTION IMPRESSION** et du sélecteur **PDFOptions**.
    - **Version** : permet de lire le numéro de version courant du pilote PDFCreator. Ce sélecteur peut être utilisé uniquement avec la commande **LIRE OPTION IMPRESSION**. Le numéro est retourné dans le paramètre *valeur1*.
    - **Last error** : permet de lire la dernière erreur retournée par le pilote PDFCreator. Ce sélecteur peut être utilisé uniquement avec la commande **LIRE OPTION IMPRESSION**. Le numéro de l'erreur est retourné dans le paramètre *valeur1*.
    - **Print in progress** : permet de savoir si 4D est en attente d'une impression de PDFCreator. Ce sélecteur peut être utilisé uniquement avec la commande **LIRE OPTION IMPRESSION**. Le paramètre *valeur1* retourne 1 si 4D est en attente de PDFCreator et 0 dans le cas contraire.
    - **Job count** : permet de connaître le nombre de tâches en attente dans la file d'impression. Ce sélecteur peut être utilisé uniquement avec la commande **LIRE OPTION IMPRESSION**. Le nombre de tâches est retourné dans le paramètre *valeur1*.
    - **Synchronous Mode** : permet de fixer le mode de synchronisation entre les requêtes d'impression envoyées par 4D et le pilote PDFCreator. Comme 4D ne peut pas obtenir d'information concernant le statut courant d'une tâche d'impression présente dans la file d'impression, une option permet de mieux contrôler l'exécution des tâches en ne les envoyant que lorsque le statut du pilote PDFCreator est "libre". Dans ce cas, 4D est synchronisé avec le pilote. Passez 0 dans *valeur1* pour que 4D envoie immédiatement les requêtes d'impression (valeur par défaut) et 1 pour que 4D soit synchronisé et attende que le pilote ait terminé sa tâche en cours avant d'en envoyer une autre.

Note : Après chaque impression, 4D rétablit automatiquement les paramètres précédents du pilote PDFCreator afin d'éviter toute interférence avec les autres programmes utilisant PDFCreator.

Exemple 1

La méthode suivante active le pilotage PDF de manière à imprimer tous les enregistrements de la table à l'emplacement C:\Test\Test_PDF_N où N est le numéro de séquence de l'enregistrement :

```
FIXER IMPRIMANTE COURANTE(PDFCreator Nom imprimante)
      // Sous Windows, sélectionner l'imprimante virtuelle installée par PDFCreator
Si(OK=1)  // Si PDFCreator est bien installé

   TOUT SELECTIONNER([Table_1])
   Boucle($i;1;Enregistrements trouves([Table_1]))
      FIXER OPTION IMPRESSION(Option destination;3;"C:\\Test\\Test_PDF_"+Chaine($i))
          // L'option de destination 3 déclenche une tâche d'impression PDFCreator
      IMPRIMER ENREGISTREMENT([Table_1];*)
      ENREGISTREMENT SUIVANT([Table_1])
   Fin de boucle
          // Réinitialisation des options du pilote PDFCreator
   FIXER OPTION IMPRESSION("PDFInfo:Reset standard options";0)
Fin de si
```

Exemple 2

La valeur de l'option Option orientation en version 64 bits peut être modifiée à l'intérieur d'une même tâche d'impression (cas particulier). A noter que l'option doit être définie avant la commande **SAUT DE PAGE** :

```
TOUT SELECTIONNER([Personnes])
PARAMETRES IMPRESSION
Si(OK=1)
   OUVRIR TACHE IMPRESSION
   FIXER OPTION IMPRESSION(Option orientation;1) //portrait
   Imprimer ligne([Personnes];"Form_Vert")
```

```
    FIXER OPTION IMPRESSION(Option orientation;2) //paysage
    SAUT DE PAGE //doit être impérativement appelé APRES l'option
    Imprimer ligne([Personnes];"Form_Hor")
    FERMER TACHE IMPRESSION
  Fin de si
```

Variables et ensembles système

La variable système OK prend la valeur 1 si la commande a été exécutée correctement, sinon elle prend la valeur 0.
Si vous passez un code d'option invalide (option non reconnue par PDFCreator par exemple), OK prend la valeur 0.

Gestion des erreurs

Si la valeur passée pour une *option* est invalide ou si elle n'est pas disponible sur l'imprimante, la commande retourne une erreur (que vous pouvez intercepter à l'aide d'une méthode de gestion d'erreur installée par la commande **APPELER SUR ERREUR**) et la valeur courante de l'option est inchangée.

FIXER TAQUET IMPRESSION

FIXER TAQUET IMPRESSION (numTaquet ; position {; *})

Paramètre	Type		Description
numTaquet	Entier long	⇒	Numéro de taquet
position	Entier long	⇒	Nouvelle position du taquet
*	Opérateur	⇒	Si passé = déplacer les marqueurs suivants Si omis = ne pas déplacer les marqueurs suivants

Description

La commande **FIXER TAQUET IMPRESSION** permet de définir la position d'un taquet au moment de l'impression. Combinée aux commandes **Lire taquet impression**, **OBJET DEPLACER** ou **Imprimer ligne**, cette commande permet d'ajuster la taille des zones d'impression.

FIXER TAQUET IMPRESSION peut être appelée dans deux contextes :

- lors de l'événement formulaire Sur entête, dans le cadre de l'utilisation des commandes **IMPRIMER SELECTION** et **IMPRIMER ENREGISTREMENT**.
- lors de l'événement formulaire Sur impression corps, dans le cadre de l'utilisation de la commande **Imprimer ligne**. Ce fonctionnement facilite l'impression d'états personnalisés (voir exemple).

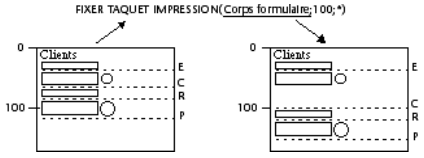
L'effet de la commande est limité à l'impression, aucune modification n'apparaît à l'écran. Les modifications apportées aux formulaires ne sont pas sauvegardées.

Passez dans le paramètre *numTaquet* une des constantes du thème "**Zone de formulaire**" :

Constante	Type	Valeur
Corps formulaire	Entier long	0
Entête formulaire	Entier long	200
Entête formulaire1	Entier long	201
Entête formulaire10	Entier long	210
Entête formulaire2	Entier long	202
Entête formulaire3	Entier long	203
Entête formulaire4	Entier long	204
Entête formulaire5	Entier long	205
Entête formulaire6	Entier long	206
Entête formulaire7	Entier long	207
Entête formulaire8	Entier long	208
Entête formulaire9	Entier long	209
Pied de page formulaire	Entier long	100
Rupture formulaire0	Entier long	300
Rupture formulaire1	Entier long	301
Rupture formulaire2	Entier long	302
Rupture formulaire3	Entier long	303
Rupture formulaire4	Entier long	304
Rupture formulaire5	Entier long	305
Rupture formulaire6	Entier long	306
Rupture formulaire7	Entier long	307
Rupture formulaire8	Entier long	308
Rupture formulaire9	Entier long	309

Passez dans *position* la nouvelle position souhaitée du taquet, exprimée en pixels.

Si vous passez le paramètre optionnel *, tous les marqueurs situés au-dessous du marqueur désigné par *numTaquet* seront déplacés du même nombre de pixels et dans la même direction que lui lors de l'exécution de la commande. **Attention :** dans ce cas, les objets éventuellement présents dans les zones situées au-dessous du marqueur sont également déplacés.
Lorsque le paramètre * est utilisé, il est donc possible de positionner le marqueur *numTaquet* au-delà de la position initiale des marqueurs qui le suivent — ces derniers étant déplacés simultanément.

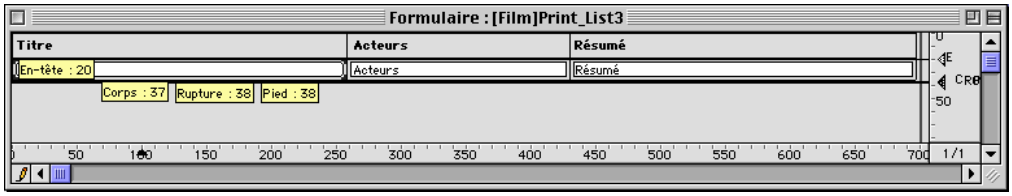


Notes :

- Cette commande modifie la position des taquets existants uniquement. Elle ne permet pas d'ajouter des taquets. Si vous désignez un taquet qui n'existe pas dans le formulaire, la commande ne fait rien.
- Le fonctionnement des taquets d'impression en mode Développement est conservé : un taquet ne peut pas aller plus haut que celui qui le précède ni plus bas que celui qui le suit (lorsque le paramètre * n'est pas utilisé).

Exemple

Cet exemple complet permet de générer l'impression d'un état sur trois colonnes, la hauteur de chaque ligne étant calculée à la volée en fonction du contenu des champs.
Le formulaire de sortie utilisé pour l'impression est le suivant :



L'événement formulaire Sur impression corps a été sélectionné pour le formulaire (rappelons que la commande **Imprimer ligne** ne génère que cet événement, quelle que soit la zone imprimée).
Pour chaque enregistrement, la hauteur de la ligne doit être adaptée en fonction du contenu de la colonne "Acteurs" ou "Résumé" (colonne ayant le plus de contenu). Voici le résultat souhaité :

Titre	Acteurs	Résumé
The Avengers	Ralph Fiennes, Uma Thurman, Sean Connery, Jim Broadbent, Patrick Macnee, Fiona Shaw, Eddie Izzard, Eileen Atkins	"The Avengers", the popular TV series from the sixties, is brought to the big screen with a mix of humour, retro fashion and action. Ralph Fiennes plays the role of well dressed John Steed and Uma Thurman is the beautiful Emma Peel dressed in a jumpsuit. Our two special agents fight crime in style. Sean Connery plays Sir August De Wynter, an evil genious that wants to take over the world with his high-tech weather machine. Unleashing snow storms or heatwaves, and armed with remote-controlled killer bees, this nut is a real menance to society. But all these climate changes won't stop our two heroes. A cup of tea, anyone?
20,000 Leagues Under The Sea		"The year is 1866, when New England's fishing harbors are the scene for a creature of unknown origin destroying ships at sea. It is the job of Professor Pierre Aronnax, a marine expert, and Ned Land, the iron-willed sailor, to learn the truth of the monster roaming the seas. The great novelist, Jules Verne, described this perilous journey to the darkest depths of the sea with Captain Nemo aboard the Nautilus."
The Adventures Of Ichabod And Mr. Toad (Walt Disney C	Bing Crosby, Basil Rathbone, Eric Blore, Pat O'Malley, John McLeish, Colin Campbell, Campbell Grant, Claud Allister	Hang on for the wild motorcar ride of J. Thaddeus Toad as he drives his friends Mole, Rat and Angus MacBadger into a worried frenzy! Then meet the spindly Ichabod Crane, who dreams of sweeping beautiful Katrina Van Tassel off her feet, despite opposition from town bully Brom Bones, who also has his eye on Katrina. The comic rivalry introduces Ichabod to the legend of the Headless Horseman resulting in a heart-thumping climax. Wonderfully narrated by Basil Rathbone and Bing Crosby, The Adventures Of Ichabod And Mr. Toad brims with high-spirited adventure, brilliant animation and captivating music you'll want to share with your family time and again.
Mission To Mars	Gary Sinise, Tim Robbins, Don Cheadle, Jerry O'Connell, Connie Nielsen	From the director of Mission Impossible comes the thrilling, eye-popping science-fiction adventure Mission To Mars - starring Gary Sinise (Snake Eyes) and Tim Robbins (Austin Powers: The Spy Who Shagged Me). The year is 2020, and the first manned mission to Mars, commanded by Luke Graham (Don Cheadle - Out Of Sight), lands safely on the red planet. But the Martian landscape harbors a bizarre and shocking secret that leads to a mysterious disaster so catastrophic, it decimates the crew. Haunted by a cryptic last message from Graham, NASA launches the Mars Recovery Mission to investigate and bring back survivors - if there are any. Confronted with nearly insurmountable dangers, but propelled by deep friendship, the team finally lands on Mars and makes a discovery so amazing, it takes your breath away. Mission To Mars is an action-packed rocket ride that will enthrall you with its stunning special effects and keep you on the edge of your seat.
The Abyss: Special Edition	Ed Harris, Mary Elizabeth Mastrantonio, Michael Biehn, Leo Burmester, Todd Graff, John Bedford Lloyd, Kimberly Scott	In this thrilling, underwater action-adventure from writer-director James Cameron (Titanic, Terminator 2: Judgment Day, Aliens), a civilian oil-rig crew is recruited to conduct a search-and-rescue effort when a nuclear submarine mysteriously sinks. One diver (Ed Harris) soon finds himself on a spectacular odyssey over 25,000 feet below the ocean's surface, where he confronts a mysterious force that has the power to change the world or destroy it. Includes both the Special Edition, with 28 minutes of additional footage, and the original theatrical version, along with the 80-minute documentary Under Pressure: Making The Abyss, and much more.
Absence Of The Good	Stephen Baldwin, Rob Knepper, Shawn Hatt, Allen Garfield, Silas Weir Mitchell, Tyne Daly	One cop. One killer. No clues. No time. After his son is shot to death at school, Detective Caleb Barnes (Stephen Baldwin) loses touch with his soul. When a series of seemingly unrelated murders plagues Salt Lake City, the detective hides his grief in search for the killer. Hampered by a lack of clues and his commander's unrelenting pressure, Caleb painstakingly unravels a tangled web, exposing a malignant family history of abuse and murder.

La méthode projet d'impression est la suivante :

```
C_ENTIER LONG(vLhauteur_imp;$vLhauteur;vLhauteur_imprimee)
C_ALPHA(31;vSimpr_zone)
UTILISER PARAMETRES IMPRESSION([Film];"Print_List3")
LIRE ZONE IMPRESSION(vLhauteur_imp)
vLhauteur_imprimee:=0
TOUT SELECTIONNER([Film])

vSimpr_zone:="Entete" `Impression de la zone d'en-tête
$vLhauteur:=Imprimer ligne([Film];"Print_List3";Entête formulaire)
$vLhauteur:=21   `Hauteur fixe
vLhauteur_imprimee:=vLhauteur_imprimee+$vLhauteur

Tant que(Non(Fin de selection([Film])))
   vSimpr_zone:="Corps" `Impression de la zone de corps
   $vLhauteur:=Imprimer ligne([Film];"Print_List3";Corps formulaire)
 `Le calcul du corps est effectué dans la méthode formulaire
   vLhauteur_imprimee:=vLhauteur_imprimee+$vLhauteur
   Si(OK=0) `NE PAS VALIDER a été exécutée dans la méthode formulaire
      SAUT DE PAGE
      vLhauteur_imprimee:=0
      vSimpr_zone:="Entete" `Réimpression de la zone d'en-tête
      $vLhauteur:=Imprimer ligne([Film];"Print_List3";Entête formulaire)
      $vLhauteur:=21
      vLhauteur_imprimee:=vLhauteur_imprimee+$vLhauteur
      vSimpr_zone:="Corps"
      $vLhauteur:=Imprimer ligne([Film];"Print_List3";Corps formulaire)
      vLhauteur_imprimee:=vLhauteur_imprimee+$vLhauteur
   Fin de si
   ENREGISTREMENT SUIVANT([Film])
```

```
Fin tant que
SAUT DE PAGE `Assurons-nous que la dernière page est imprimée
```

La méthode du formulaire Print_List3 est la suivante :

```
C_ENTIER LONG($g;$h;$d;$b;$larg_fix;$haut_préc;$g1;$h1;$d1;$b1)
C_ENTIER LONG($pos_finale;$i)
C_ENTIER LONG($position_c;$position_e;$hauteur_a_imprimer;$hauteur_restante)

Au cas ou
   :(vSimpr_zone="Corps") `Impression du corps en cours
      OBJET LIRE COORDONNEES([Film]Acteurs;$g;$h;$d;$b)
      $larg_fix:=$d-$g  `Calcul de la taille du champ texte Acteurs
      $haut_préc:=$b-$h
      OBJET LIRE TAILLE OPTIMALE([Film]Acteurs;$larg;$haut;$larg_fix)
 `Taille optimale du champ en fonction du contenu
      $deplacement:=$haut-$haut_préc

      OBJET LIRE COORDONNEES([Film]Résumé;$g1;$h1;$d1;$b1)
      $larg_fix1:=$d1-$g1  `Calcul de la taille du champ texte Résumé
      $haut_préc1:=$b1-$h1
      OBJET LIRE TAILLE OPTIMALE([Film]Résumé;$larg1;$haut1;$larg_fix1)
 `Taille optimale du champ en fonction du contenu
      $deplacement1:=$haut1-$haut_préc1
      Si($deplacement1>$deplacement)
 `On détermine le champ le plus haut
         $deplacement:=$deplacement1
      Fin de si

      Si($deplacement>0)
         $position:=Lire taquet impression(Corps formulaire)
         $pos_finale:=$position+$deplacement
 `On déplace le taquet Corps et ceux qui le suivent
         FIXER TAQUET IMPRESSION(Corps formulaire;$pos_finale;*)
 `Redimensionnement des zones de texte
         OBJET DEPLACER([Film]Acteurs;$g;$h;$d;$haut+$h;*)
         OBJET DEPLACER([Film]Résumé;$g1;$h1;$d1;$haut1+$h1;*)

 `Redimensionnement des lignes de séparation
         OBJET LIRE COORDONNEES(*;"LigneH1";$g;$h;$d;$b)
         OBJET DEPLACER(*;"LigneH1";$g;$pos_finale-1;$d;$pos_finale;*)
         Boucle($i;1;4;1)
            OBJET LIRE COORDONNEES(*;"LigneV"+Chaine($i);$g;$h;$d;$b)
            OBJET DEPLACER(*;"LigneV"+Chaine($i);$g;$h;$d;$pos_finale;*)
         Fin de boucle
      Fin de si

 `Calcul de la place disponible
      $position_c:=Lire taquet impression(Corps formulaire)
      $position_e:=Lire taquet impression(Entête formulaire)
      $hauteur_a_imprimer:=$position_c-$position_e
      $hauteur_restante:=hauteur_impression-vLhauteur_imprimee
      Si($hauteur_restante<$hauteur_a_imprimer) `Hauteur insuffisante
         NE PAS VALIDER `Passer la ligne sur la page suivante
      Fin de si
Fin de cas
```

IMPRIMER ENREGISTREMENT

IMPRIMER ENREGISTREMENT ({laTable}{;}{* | >})

Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle imprimer l'enregistrement courant ou Table par défaut si ce paramètre est omis
* | >	Opérateur	⇒	* pour supprimer les boîtes de dialogue d'impression ou > pour ne pas réinitialiser les paramètres d'impression

Description

Cette commande provoque l'impression de l'enregistrement courant de *laTable*, sans modifier la sélection courante. Le formulaire sortie courant est utilisé pour l'impression. S'il n'y a pas d'enregistrement courant dans *laTable*, **IMPRIMER ENREGISTREMENT** ne fait rien.

IMPRIMER ENREGISTREMENT permet d'imprimer des sous-formulaires, ce qui n'est pas possible avec la commande **Imprimer ligne**.

Note : Si l'enregistrement a subi des modifications qui n'ont pas encore été sauvegardées sur disque, la commande imprime les valeurs les plus récentes, et non celles stockées sur le disque.

Par défaut, **IMPRIMER ENREGISTREMENT** affiche les deux boîtes de dialogue de paramétrage d'impression (4D version 32 bits) ou la boîte de dialogue d'impression (4D version 64 bits). Si l'utilisateur annule une des deux boîtes de dialogue, l'exécution de la commande est stoppée.
Vous pouvez supprimer leur affichage en utilisant soit le paramètre optionnel astérisque (*), soit le paramètre optionnel "supérieur à" (>).

- Le paramètre * provoque une impression avec les paramètres d'impression courants (paramètres par défaut ou définis par les commandes **UTILISER PARAMETRES IMPRESSION** et/ou **FIXER OPTION IMPRESSION**).
- Le paramètre > provoque en outre l'impression sans réinitialisation des paramètres d'impression. Ce paramètre est utile lorsque vous souhaitez exécuter successivement plusieurs appels à **IMPRIMER ENREGISTREMENT** (par exemple à l'intérieur d'une boucle) tout en conservant des paramètres d'impression personnalisés préalablement définis.

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :

- Veillez à ce qu'aucune boîte de dialogue n'apparaisse sur le poste serveur (sauf besoin spécifique). Pour cela, il est nécessaire d'appeler la commande avec le paramètre * ou >.
- En cas de problème sur l'imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Attention : N'utilisez pas la commande **SAUT DE PAGE** avec **IMPRIMER ENREGISTREMENT**. **SAUT DE PAGE** est exclusivement réservée à une utilisation combinée avec la commande **Imprimer ligne**.

Exemple 1

L'exemple suivant imprime l'enregistrement courant de la table *[Factures]*. Cette méthode est celle d'un bouton **Imprimer** placé dans le formulaire entrée. Lorsque l'utilisateur clique sur ce bouton, l'enregistrement est imprimé dans un formulaire spécialement créé dans ce but.

```
FORM FIXER SORTIE([Factures];"ImpressionEnregistrement") `Sélection du formulaire pour l'impression
IMPRIMER ENREGISTREMENT([Factures];*) `Imprimer les factures (sans dialogues d'impression)
FORM FIXER SORTIE([Factures];"FormListe") `Restauration du formulaire sortie courant
```

Exemple 2

L'exemple suivant imprime le même enregistrement courant dans deux formulaires différents. Cette méthode est celle d'un bouton **Imprimer** placé dans un formulaire entrée. Vous souhaitez définir des paramètres d'impression personnalisés et les utiliser pour les deux formulaires.

```
PARAMETRES IMPRESSION `L'utilisateur définit ses paramètres d'impression
Si(OK=1)
   FORM FIXER SORTIE([Employés];"Détaillé") `Utiliser un premier formulaire d'impression
   IMPRIMER ENREGISTREMENT([Employés];>)
 `Imprimer en utilisant les paramètres définis par l'utilisateur
   FORM FIXER SORTIE([Employés];"Simplifié") `Utiliser un second formulaire d'impression
   IMPRIMER ENREGISTREMENT([Employés];>)
 `Imprimer en utilisant les paramètres définis par l'utilisateur
   FORM FIXER SORTIE([Employés];"Sortie") `Rétablir le formulaire sortie par défaut
Fin de si
```

IMPRIMER ETIQUETTES

IMPRIMER ETIQUETTES ({laTable }{;}{ nomFichier {; * | >}})

Paramètre	Type		Description
laTable	Table	⇒	Table à imprimer ou Table par défaut si ce paramètre est omis
nomFichier	Chaîne	⇒	Nom de fichier d'étiquettes sur disque
* | >		⇒	* pour supprimer les boîtes de dialogue d'impression ou > pour ne pas réinitialiser les paramètres d'impression

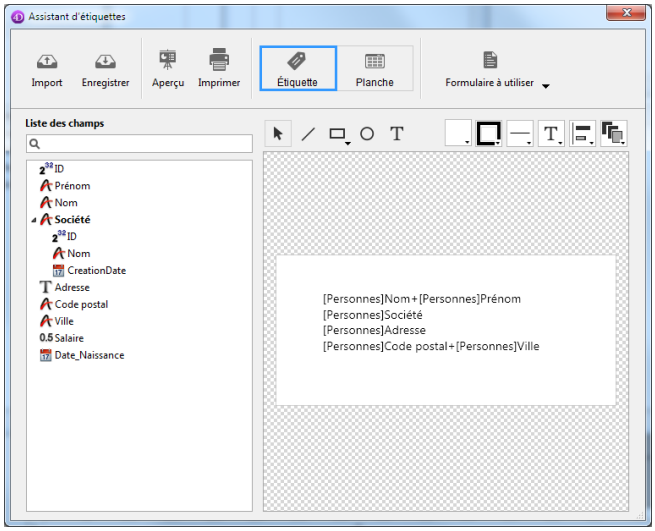
Description

IMPRIMER ETIQUETTES vous permet d'imprimer des étiquettes à partir des données de la sélection de *laTable*.

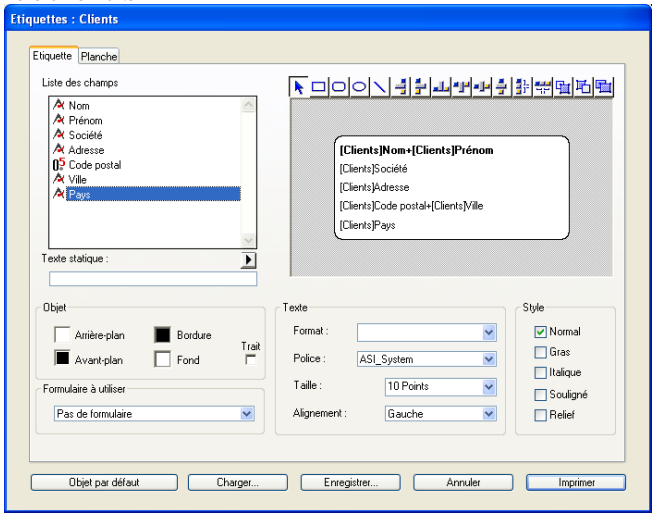
Si vous ne spécifiez pas le paramètre *nomFichier*, **IMPRIMER ETIQUETTES** imprime la sélection courante de *laTable* sous forme d'étiquettes, en utilisant le formulaire sortie courant du process. Vous ne pouvez pas imprimer de sous-formulaires lorsque vous utilisez cette commande. Pour plus d'informations sur la création d'étiquettes à l'aide de formulaires, reportez-vous au manuel *Mode Développement* de 4D.

Si vous spécifiez le paramètre *nomFichier*, **IMPRIMER ETIQUETTES** vous permet d'imprimer un document d'étiquettes existant stocké sur disque ou d'ouvrir l'Assistant de création d'étiquettes (affiché ci-dessous). Pour plus d'informations sur ce point, reportez-vous à l'exemple plus bas.

Version 64 bits :



Version 32 bits :



Note de compatibilité : L'éditeur d'étiquettes 32 bits prend uniquement en charge le jeu de caractères ASCII. Si vous avez besoin d'utiliser un jeu de caractères plus étendu, vous devez imprimer les étiquettes via le formulaire sortie courant.

Par défaut, **IMPRIMER ETIQUETTES** affiche les deux boîtes de dialogue de paramétrage d'impression (4D version 32 bits) ou la boîte de dialogue d'impression (4D version 64 bits). Si l'utilisateur annule une des deux boîtes de dialogue, l'exécution de la commande est stoppée et l'état n'est pas imprimé. Vous pouvez supprimer leur affichage en utilisant soit le paramètre optionnel astérisque (*), soit le paramètre optionnel “supérieur à” (>).

- Le paramètre * provoque une impression avec les paramètres d'impression courants.
- Le paramètre > provoque en outre l'impression sans réinitialisation des paramètres d'impression. Ce paramètre est utile lorsque vous souhaitez exécuter successivement plusieurs appels à **IMPRIMER ETIQUETTES** (par exemple à l'intérieur d'une boucle) tout en conservant des paramètres d'impression personnalisés préalablement définis.

Il est à noter que ces paramètres n'ont pas d'effet si l'assistant de création d'étiquettes est utilisé.

Si l'assistant de création d'étiquettes n'est pas utilisé, la variable système OK est mise à 1 si toutes les étiquettes ont été imprimées ; sinon, elle prend la valeur 0 (zéro) (par exemple si l'utilisateur a cliqué sur le bouton **Annuler** dans les boîtes de dialogue d'impression).

Si vous spécifiez le paramètre *nomFichier*, les étiquettes sont imprimées avec les paramétrages définis dans *nomFichier*. Si *nomFichier* est une chaîne vide (""), **IMPRIMER ETIQUETTES** affiche une boîte de dialogue standard d'ouverture de documents, permettant à l'utilisateur de sélectionner le fichier d'étiquettes à utiliser. Si *nomFichier* est le nom d'un fichier qui n'existe pas ou est invalide (si vous passez, par exemple, **Caractere(1)** dans *nomFichier*),

l'assistant de création d'étiquettes s'affiche, permettant à l'utilisateur de créer son propre format d'étiquettes.

Note : Si *laTable* a été déclarée “invisible” en mode Développement, l'assistant de création d'étiquettes n'apparaît pas.

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :

- Veillez à ce qu’aucune boîte de dialogue n’apparaisse sur le poste serveur (sauf besoin spécifique). Pour cela, il est nécessaire d’appeler la commande avec le paramètre * ou >.
- La syntaxe faisant apparaître l‘éditeur d’étiquettes ne fonctionne pas avec 4D Server, dans ce cas, la variable système OK prend la valeur 0.
- En cas de problème sur l’imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Exemple 1

L'exemple suivant imprime des étiquettes à l'aide du formulaire de sortie de la table. L'exemple s'appuie sur deux méthodes. La première est une méthode projet qui désigne le formulaire sortie à utiliser puis imprime les étiquettes :

```
TOUT SELECTIONNER([Adresses]) ` Sélection de tous les enregistrements
FORM FIXER SORTIE([Adresses];"ImprimEtiq") ` Sélection du formulaire sortie
IMPRIMER ETIQUETTES([Adresses]) ` Impression des étiquettes
FORM FIXER SORTIE([Adresses];"Sortie") ` Rétablissement du formulaire sortie par défaut
```

La seconde méthode est la méthode du formulaire "ImprimEtiq". Le formulaire contient une variable, nommée *vEtiq*, contenant les champs concaténés. Si le second champ Adresse (Adr2) est vide, il est enlevé par la méthode (notez que cette opération peut être effectuée automatiquement par l'assistant de création d'étiquettes). La méthode formulaire construit l'étiquette pour chaque enregistrement :

```
  ` Méthode formulaire [Adresses];"Etiquette sortie"
Au cas ou
   :(Evenement formulaire=Sur chargement)
      vEtiq:=[Adresses]Nom1+" "+[Adresses]Nom2+Caractere(13)+[Adresses]Adr1
      +Caractere(13)
      Si([Adresses]Adr2 #"")
         vEtiq:=vEtiq+[Adresses]Adr2+Caractere(13)
      Fin de si
      vEtiq:=vEtiq+[Adresses]Ville+", "+[Adresses]Département+" "+[Adresses]Code Postal
Fin de cas
```

Exemple 2

L'exemple suivant permet à l'utilisateur d'effectuer une recherche sur la table [Employés], et d'imprimer automatiquement les étiquettes “Mes Etiquettes” :

```
CHERCHER([Employés])
Si(OK=1)
   IMPRIMER ETIQUETTES([Employés];"Mes Etiquettes";*)
Fin de si
```

Exemple 3

L'exemple suivant permet à l'utilisateur d'effectuer une recherche sur la table [Employés], puis de choisir les étiquettes qui doivent être imprimées :

```
CHERCHER([Employés])
Si(OK=1)
   IMPRIMER ETIQUETTES([Employés];"")
Fin de si
```

Exemple 4

L'exemple suivant permet à l'utilisateur d'effectuer une recherche sur la table [Employés], puis affiche l'assistant de création d'étiquettes afin que l'utilisateur puisse concevoir, sauvegarder, charger et imprimer tout type d'étiquettes :

```
CHERCHER([Employés])
Si(OK=1)
   IMPRIMER ETIQUETTES([Employés];Caractere(1))
Fin de si
```

Imprimer ligne

Imprimer ligne ({laTable ;} formulaire {; zone1 {; zone2}}) -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table à imprimer, ou Table par défaut si ce paramètre est omis
formulaire	Chaîne	⇒	Formulaire à imprimer
zone1	Entier long	⇒	Marqueur d'impression, ou Zone de départ (si zone2 est spécifié)
zone2	Entier long	⇒	Zone de fin (si zone1 est spécifié)
Résultat	Entier long	⇐	Hauteur de la section imprimée

Description

La commande **Imprimer ligne** imprime simplement *formulaire* avec les valeurs courantes des champs et des variables de *laTable*. Cette commande est généralement utilisée pour imprimer des états particulièrement complexes nécessitant un contrôle total du processus d'impression. **Imprimer ligne** ne gère pas les traitements d'enregistrements, ni les ruptures, sauts de pages, en-têtes ou pieds de pages. Vous devez vous-même prendre en charge ces opérations. **Imprimer ligne** imprime uniquement des champs et des variables avec une taille fixe, la commande ne gère pas les objets de taille variable.

Comme la commande **Imprimer ligne** ne génère pas de saut de page après avoir imprimé un formulaire, elle vous permet de combiner facilement différents formulaires sur la même page. Ainsi, **Imprimer ligne** est idéale pour effectuer des impressions complexes impliquant plusieurs tables et plusieurs formulaires. Pour "forcer" 4D à effectuer un saut de page entre deux formulaires, utilisez la commande **SAUT DE PAGE**. Pour reporter sur la page suivante l'impression d'un formulaire dont la hauteur est supérieure à la place disponible, appelez la commande **NE PAS VALIDER** avant la commande **SAUT DE PAGE**.

Trois syntaxes peuvent être utilisées :

- **Impression du corps d'un formulaire**

Syntaxe :

```
hauteur:=Imprimer ligne(maTable;monFormulaire)
```

Dans ce cas, **Imprimer ligne** imprime uniquement la zone de Corps du formulaire (zone située entre les marqueurs En-tête et Corps).

- **Impression de zone de formulaire**

Syntaxe :

```
hauteur:=Imprimer ligne(maTable;monFormulaire;marqueur)
```

Dans ce cas, la commande imprimera la section désignée par *marqueur*. Passez dans le paramètre *marqueur* une des constantes du thème "**Zone de formulaire**" :

Constante	Type	Valeur
Corps formulaire	Entier long	0
Entête formulaire	Entier long	200
Entête formulaire1	Entier long	201
Entête formulaire10	Entier long	210
Entête formulaire2	Entier long	202
Entête formulaire3	Entier long	203
Entête formulaire4	Entier long	204
Entête formulaire5	Entier long	205
Entête formulaire6	Entier long	206
Entête formulaire7	Entier long	207
Entête formulaire8	Entier long	208
Entête formulaire9	Entier long	209
Pied de page formulaire	Entier long	100
Rupture formulaire0	Entier long	300
Rupture formulaire1	Entier long	301
Rupture formulaire2	Entier long	302
Rupture formulaire3	Entier long	303
Rupture formulaire4	Entier long	304
Rupture formulaire5	Entier long	305
Rupture formulaire6	Entier long	306
Rupture formulaire7	Entier long	307
Rupture formulaire8	Entier long	308
Rupture formulaire9	Entier long	309

- **Impression de section**

Syntaxe :

```
hauteur:=Imprimer ligne(maTable;monFormulaire;zoneDébut;zoneFin)
```

Dans ce cas, la commande imprimera la section comprise entre les paramètres *zoneDébut* et *zoneFin*. Les valeurs saisies doivent être exprimées en pixels.

La valeur retournée par **Imprimer ligne** indique la hauteur de la zone d'impression. Cette valeur sera automatiquement prise en compte par la commande **Lire hauteur imprimee**.

Les boîtes de dialogue standard d'impression n'apparaissent pas lorsque vous utilisez la commande **Imprimer ligne**. L'état généré ne tient pas compte des paramètres d'impression définis en mode Développement pour le formulaire. Il y a deux manières de définir les paramètres d'impression avant d'effectuer une série d'appels à **Imprimer ligne** :

- Vous appelez **PARAMETRES IMPRESSION**. Dans ce cas, vous laissez l'utilisateur définir ses paramètres dans les boîtes de dialogue d'impression.
- Vous appelez **UTILISER PARAMETRES IMPRESSION**. Dans ce cas, les paramètres sont définis par programmation.

Imprimer ligne construit chaque page à imprimer en mémoire. La page n'est imprimée que lorsqu'elle est entièrement remplie ou lorsque vous appelez **SAUT DE PAGE**. Pour vous assurer que la dernière page d'une impression exécutée par l'intermédiaire **Imprimer ligne** soit effectivement imprimée, vous devez conclure par un appel à la commande **SAUT DE PAGE** (hormis dans le contexte d'un **OUVRIR TACHE IMPRESSION**, cf. note). Sinon, la dernière page, si elle n'est pas pleine, reste en mémoire et n'est pas imprimée.

Attention : Si la commande est appelée dans le contexte d'une tâche d'impression ouverte avec **OUVRIR TACHE IMPRESSION**, vous ne devez PAS appeler **SAUT DE PAGE** pour la dernière page car celle-ci est automatiquement imprimée par la commande **FERMER TACHE IMPRESSION**. Si vous appelez **SAUT DE PAGE** dans ce cas, une page vide est imprimée.

Cette commande imprime les zones et objets externes (par exemple des zones 4D Write ou 4D View). La zone est réinitialisée à chaque exécution de la commande.

Attention : Imprimer ligne n'imprime pas les sous-formulaires. Si vous voulez imprimer uniquement un formulaire comportant de tels objets, utilisez plutôt **IMPRIMER ENREGISTREMENT**.

Imprimer ligne ne génère qu'un événement Sur impression corps pour la méthode formulaire.

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :

- Veillez à ce qu'aucune boîte de dialogue n'apparaisse sur le poste serveur (sauf besoin spécifique).
- En cas de problème sur l'imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Exemple 1

L'exemple suivant effectue la même chose que ce que ferait la commande **IMPRIMER SELECTION**. Cependant, l'état utilise deux formulaires différents suivant le type d'enregistrement (chèque émis ou dépôt) :

```
CHERCHER([Opérations]) ` Permettre à l'utilisateur de sélectionner les enregistrements
Si(OK=1)
   TRIER([Opérations]) ` Permettre à l'utilisateur de trier les enregistrements
   Si(OK=1)
      PARAMETRES IMPRESSION ` Afficher les boîtes de dialogue d'impression
      Si(OK=1)
         Boucle($i;1;Enregistrements trouves([Opérations]))
            Si([Opérations]Type="Chèque") ` Si c'est un chèque…
               Imprimer ligne([Opérations];"SortieChèque") ` Utiliser un formulaire de chèque
            Sinon ` Sinon c'est un dépôt donc…
               Imprimer ligne([Opérations];"SortieDépôt") ` Utiliser un formulaire de dépôt...
            Fin de si
            ENREGISTREMENT SUIVANT([Opérations])
         Fin de boucle
         SAUT DE PAGE `S'assurer que la dernière page est imprimée
      Fin de si
   Fin de si
Fin de si
```

Exemple 2

Reportez-vous à l'exemple de la commande **FIXER TAQUET IMPRESSION**.

Imprimer objet

Imprimer objet ({* ;} objet {; posX {; posY {; largeur {; hauteur}}}}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Variable (si * omis)
posX	Entier long	⇒	Emplacement horizontal de l'objet
posY	Entier long	⇒	Emplacement vertical de l'objet
largeur	Entier long	⇒	Largeur de l'objet (pixels)
hauteur	Entier long	⇒	Hauteur de l'objet (pixels)
Résultat	Booléen	⇐	Vrai = objet entièrement imprimé, Faux sinon

Description

La commande **Imprimer objet** vous permet d'imprimer le ou les objet(s) de formulaire désigné(s) par les paramètres *objet* et *, à l'emplacement défini par les paramètres *posX* et *posY*.

Avant d'appeler la commande **Imprimer objet**, vous devez désigner le formulaire table ou projet contenant les objets à imprimer, à l'aide de la commande **FORM CHARGER**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne de caractères). Si vous ne passez pas le paramètre *, vous indiquez que *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable (de type objet uniquement).

Les paramètres *posX* et *posY* définissent le point de départ de l'impression du ou des objet(s). Ces valeurs doivent être exprimées en pixels. Si ces paramètres sont omis, l'objet sera imprimé en fonction de son emplacement dans le formulaire.

Les paramètres *largeur* et *hauteur* vous permettent de définir la largeur et la hauteur de l'objet de formulaire. La commande **Imprimer objet** ne gère pas les objets de taille variable. Vous devez utiliser la commande **OBJET LIRE TAILLE OPTIMALE** pour prendre en charge la taille des objets. Vous pouvez également utiliser la commande **OBJET LIRE TAILLE OPTIMALE** pour connaître la taille la plus adéquate pour les objets contenant du texte. De même, **Imprimer objet** ne provoque pas de sauts de page automatiques. Vous devez les gérer en fonction de vos besoins.

Vous pouvez utiliser les commandes de 4D pour modifier à la volée les propriétés des objets (couleur, taille...).

La commande retourne Vrai si l'objet a été imprimé entièrement et Faux dans le cas contraire, c'est-à-dire si l'intégralité des données associées à l'objet n'a pas pu être imprimée dans le cadre imposé. Typiquement, la commande retourne Faux lors de l'impression d'une list box, si toutes les lignes de la list box n'ont pas pu être imprimées. Dans ce cas, il suffit d'appeler la commande **Imprimer objet** de façon répétée, jusqu'à ce qu'elle retourne Vrai : un mécanisme spécifique provoque automatiquement le défilement du contenu de l'objet après chaque appel.

Notes :

- Dans la version actuelle de 4D, seuls les objets de type list box bénéficient de ce mécanisme (la commande retourne toujours Vrai pour tous les autres types d'objets). Dans les prochaines versions de 4D, ce fonctionnement sera étendu à d'autres objets au contenu variable.
- La commande **LISTBOX LIRE INFORMATION IMPRESSION** permet de contrôler le statut de l'impression durant l'opération.

La commande **Imprimer objet** peut être utilisée uniquement dans le contexte d'une tâche d'impression préalablement ouverte avec la commande **OUVRIR TACHE IMPRESSION**. Si elle n'est pas appelée dans ce contexte, la commande ne fait rien. Plusieurs commandes **Imprimer objet** peuvent être appelées dans la même tâche d'impression.

Note : Les listes hiérarchiques, les sous-formulaires et les zones Web ne sont pas imprimables.

Exemple 1

Exemple d'impression de dix objets dans un formulaire :

```
PARAMETRES IMPRESSION
Si(OK=1)
      OUVRIR TACHE IMPRESSION
      Si(OK=1)
         FORM CHARGER("PrintForm")
         x:=100
         y:=50
         LIRE ZONE IMPRESSION(hpaper;wpaper)
         Boucle($i;1;10)
            OBJET LIRE TAILLE OPTIMALE(*;"Obj"+Chaine($i);bestwidth;bestheight)
            $fin:=Imprimer objet(*;"Obj"+Chaine($i))
            y:=y+bestheight+15
            Si(y>hpaper)
               SAUT DE PAGE(>)
               y:=50
            Fin de si
         Fin de boucle
      Fin de si
      FERMER TACHE IMPRESSION
Fin de si
```

Exemple 2

Exemple d'impression d'une list box complète :

```
Repeter
      $fin:=Imprimer objet(*;"malistbox")
Jusque($fin)
```

IMPRIMER SELECTION

IMPRIMER SELECTION ({laTable}{;}{* | >})

Paramètre	Type		Description
laTable	Table	⇒	Table à laquelle appartient la sélection à imprimer ou Table par défaut si ce paramètre est omis
* | >	Opérateur	⇒	* pour supprimer les boîtes de dialogue d'impression ou > pour ne pas réinitialiser les paramètres d'impression

Description

La commande **IMPRIMER SELECTION** imprime la sélection courante de *laTable*. Les enregistrements sont imprimés dans le formulaire sortie courant de la table du process en cours. **IMPRIMER SELECTION** a le même effet que la commande **Imprimer...** du mode Développement. Si la sélection courante est vide, **IMPRIMER SELECTION** ne fait rien.

Par défaut, **IMPRIMER SELECTION** affiche les deux boîtes de dialogue de paramétrage d'impression (4D version 32 bits) ou la boîte de dialogue d'impression (4D version 64 bits). Si l'utilisateur annule une des deux boîtes de dialogue, l'exécution de la commande est stoppée et l'état n'est pas imprimé. Vous pouvez supprimer leur affichage en utilisant soit le paramètre optionnel astérisque (*), soit le paramètre optionnel “supérieur à” (>).

- Le paramètre * provoque une impression avec les paramètres d'impression courants (paramètres par défaut ou définis par les commandes **UTILISER PARAMETRES IMPRESSION** et/ou **FIXER OPTION IMPRESSION**).
- Le paramètre > provoque en outre l’impression sans réinitialisation des paramètres d’impression. Ce paramètre est utile lorsque vous souhaitez exécuter successivement plusieurs appels à **IMPRIMER SELECTION** (par exemple à l'intérieur d’une boucle) tout en conservant des paramètres d’impression personnalisés préalablement définis.

Pendant l'impression, la méthode du formulaire sortie et les méthodes objet du formulaire sont exécutées en fonction des événements sélectionnés dans les propriétés des formulaires et des objets, en mode Développement, ainsi que des événements effectivement générés :

- Un événement Sur entête est généré juste avant que la zone d'en-tête soit imprimée.
- Un événement Sur impression corps est généré juste avant que l'enregistrement soit imprimé.
- Un événement Sur impression sous total est généré juste avant qu'une zone de rupture soit imprimée.
- Un événement Sur impression pied de page est généré juste avant que la zone de pied de page soit imprimée.

Vous pouvez savoir si **IMPRIMER SELECTION** est sur le point d'imprimer le premier en-tête en testant **Avant selection** pendant un événement Sur entête. Vous pouvez également savoir si **IMPRIMER SELECTION** est sur le point d'imprimer le dernier pied de page, en testant **Fin de selection** pendant un événement Sur impression pied de page.
Pour plus d'informations, reportez-vous à la description de ces commandes ainsi qu'aux commandes **Evenement formulaire** et **Niveau**.

Si **IMPRIMER SELECTION** est appelée au même moment par deux process différents, l'impression déclenchée par le second process attendra que le premier ait terminé.

Pour imprimer une sélection triée avec des sous-totaux ou des ruptures à l'aide de la commande **IMPRIMER SELECTION**, vous devez d'abord trier la sélection. Puis vous devez inclure, dans chaque zone de rupture de l'état, une variable associée à une méthode objet assignant le sous-total à la variable. Vous pouvez aussi utiliser des fonctions statistiques ou arithmétiques telles que **Somme** et **Moyenne** pour assigner des valeurs aux variables. Pour plus d'informations, reportez-vous à la description des commandes **Sous total**, **NIVEAUX DE RUPTURES** et **CUMULER SUR**.

Attention : N'utilisez pas la commande **SAUT DE PAGE** avec **IMPRIMER SELECTION**. **SAUT DE PAGE** est exclusivement réservée à une utilisation combinée avec la commande **Imprimer ligne**.

Après un appel à **IMPRIMER SELECTION**, la variable OK prend la valeur 1 si l'impression s'est déroulée correctement. Si l'impression a été interrompue (par exemple l'utilisateur a cliqué sur un bouton Annuler dans les boîtes de dialogue d'impression), la variable OK prend la valeur *0* (zéro).

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :

- Veillez à ce qu’aucune boîte de dialogue n’apparaisse sur le poste serveur (sauf besoin spécifique). Pour cela, il est nécessaire d’appeler la commande avec le paramètre * ou >.
- En cas de problème sur l’imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Exemple

L'exemple suivant sélectionne la totalité des enregistrements de la table [Personnes]. La commande **VISUALISER SELECTION** est alors appelée pour afficher les enregistrements et permettre à l'utilisateur de sélectionner ceux qu'il souhaite imprimer. Enfin, les enregistrements choisis sont récupérés à l'aide de la commande **UTILISER ENSEMBLE** et imprimés par **IMPRIMER SELECTION** :

```
TOUT SELECTIONNER([Personnes]) ` Sélection de tous les enregistrements
VISUALISER SELECTION([Personnes];*) ` Affichage des enregistrements
UTILISER ENSEMBLE("UserSet") ` Utiliser uniquement les enregistrements sélectionnés par l'utilisateur
IMPRIMER SELECTION([Personnes]) ` Imprimer les enregistrements sélectionnés
```

Lire apercu impression

Lire apercu impression -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai = Impression à l'écran, Faux = Pas d'impression écran

Description

La commande **Lire apercu impression** retourne Vrai si l'instruction **FIXER APERCU IMPRESSION** a été appelée avec la valeur **Vrai** dans le process courant.

A noter que l'utilisateur peut modifier cette option avant de valider la boîte de dialogue. Pour obtenir le mode final d'impression, vous devez utiliser la commande **Est en apercu impression**.

Lire hauteur imprimee

Lire hauteur imprimee -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Position du marqueur

Description

La commande **Lire hauteur imprimee** retourne la hauteur globale (en pixels) de la section imprimée par la commande **Imprimer ligne**.

La valeur retournée sera comprise entre 0 (le bord supérieur de la page) et la hauteur globale retournée par la commande **LIRE ZONE IMPRESSION** (la taille maximum de la zone d'impression).

Si vous imprimez une nouvelle section via la commande **Imprimer ligne**, la hauteur de cette section est ajoutée à cette valeur. Si la zone d'impression disponible est insuffisante pour contenir cette section, une nouvelle page est générée et la valeur retournée est 0.

Les marges d'impression gauche et droite n'influent pas sur la valeur retournée, à la différence des marges inférieure et supérieure (définies éventuellement via la commande **FIXER MARGE IMPRESSION**).

Lire imprimante courante

Lire imprimante courante -> Résultat

Paramètre	Type		Description
Résultat	Chaîne	➲	Nom de l'imprimante courante

Description

La commande **Lire imprimante courante** retourne le nom de l'imprimante courante définie dans l'application 4D. Par défaut au lancement de 4D, l'imprimante courante est l'imprimante définie dans le système.

Si l'imprimante courante est gérée via un serveur d'impression ("spouleur"), le chemin d'accès complet (sous Windows) ou le nom du spouleur (sous Mac OS) est retourné.

Pour obtenir la liste des imprimantes disponibles ainsi que des informations complémentaires, utilisez la commande **LISTE IMPRIMANTES**. Pour modifier l'imprimante courante, utilisez la commande **FIXER IMPRIMANTE COURANTE**.

Note : Lorsque la constante Driver PDF générique est utilisée avec **FIXER IMPRIMANTE COURANTE**, **Lire imprimante courante** retourne "_4d_pdf_printer" ou le véritable nom du pilote PDF, le cas échéant (option non disponible avec 4D 32 bits).

Variables et ensembles système

Si aucune imprimante n'est installée, la variable système OK prend la valeur 0. Sinon, OK prend la valeur 1.

LIRE MARGE IMPRESSION

LIRE MARGE IMPRESSION (gauche ; haut ; droite ; bas)

Paramètre	Type		Description
gauche	Entier long	⇦	Marge gauche
haut	Entier long	⇦	Marge supérieure
droite	Entier long	⇦	Marge droite
bas	Entier long	⇦	Marge inférieure

Description

La commande **LIRE MARGE IMPRESSION** retourne les valeurs courantes des différentes marges définies lors de l'utilisation des commandes **Imprimer ligne**, **IMPRIMER SELECTION** et **IMPRIMER ENREGISTREMENT**.

Les valeurs sont retournées en pixels par rapport au bord du papier.

Il est possible d'obtenir la taille du papier à l'aide de la fonction **LIRE ZONE IMPRESSION**, et ainsi de calculer la zone imprimable.

Gestion des marges d'impression

Par défaut, dans 4D le calcul des impressions est effectué sur la base des "marges imprimante". L'avantage de ce système est que les formulaires s'adaptent automatiquement aux nouvelles imprimantes (puisque positionnés dans la partie imprimable). En revanche, dans le cas des formulaires pré-imprimés, il n'est pas possible de positionner précisément les éléments à imprimer car un changement d'imprimante peut modifier les marges imprimante.

Il est possible de baser l'impression des formulaires effectuée à l'aide des commandes **Imprimer ligne**, **IMPRIMER SELECTION** et **IMPRIMER ENREGISTREMENT** sur une marge fixe et identique sur chaque imprimante : la marge papier, c'est-à-dire les limites physiques de la feuille. Pour cela, il suffit d'utiliser les commandes **LIRE MARGE IMPRESSION**, **FIXER MARGE IMPRESSION** et **LIRE ZONE IMPRESSION**.

Terminologie des impressions

- **Marge papier** : la marge papier correspond aux limites physiques de la feuille.
- **Marge imprimante** : la marge imprimante est la marge au-dela de laquelle l'imprimante est incapable d'imprimer (pour des raisons physiques : galets d'impression, fin de course de la tête d'impression...). Elle varie d'une imprimante à l'autre et d'un format à l'autre.
- **Marge morte** : c'est la zone située entre la marge papier et la marge imprimante.

LIRE OPTION IMPRESSION

LIRE OPTION IMPRESSION (option ; valeur1 {; valeur2})

Paramètre	Type		Description
option	Entier long	⇨	Numéro d'option ou Code d'option PDF
valeur1	Entier long, Texte	⇦	Valeur 1 de l'option
valeur2	Entier long, Texte	⇦	Valeur 2 de l'option

Description

La commande **LIRE OPTION IMPRESSION** retourne la ou les valeur(s) courante(s) d'une option d'impression.

Le paramètre *option* vous permet de désigner l'option à lire. Vous pouvez passer soit une option standard (entier long), soit un code d'option PDF (chaîne). La commande retourne dans les paramètres *valeur1* et (facultativement) *valeur2* la ou les valeur(s) courante(s) de l'*option* spécifiée.

Pour spécifier une option d'impression standard, utilisez l'une des constantes suivantes du thème **Options d'impression** :

Constante	Type	Valeur	Comment
Option papier	Entier long	1	Si vous passez uniquement *valeur1*, il contient le nom du papier. Si vous passez les deux paramètres, *valeur1* contient la largeur du papier et *valeur1* contient la hauteur du papier. La largeur et la hauteur sont exprimées en pixels écran. Utilisez la commande **VALEURS OPTION IMPRESSION** pour connaître le nom, la hauteur et la largeur de tous les formats de papier proposés par l'imprimante.
Option orientation	Entier long	2	*valeur1* uniquement : 1=Portrait, 2=Paysage. Si une option d'orientation différente est utilisée, **LIRE OPTION IMPRESSION** retourne 0 dans *valeur1*. **Versions 64 bits** : Cette option peut être appelée au sein d'une tâche d'impression, ce qui signifie que vous pouvez passer du mode portrait au mode paysage et inversement dans la même tâche d'impression.
Option échelle	Entier long	3	*valeur1* uniquement : valeur d'échelle en pourcentage. Attention, certaines imprimantes ne permettent pas de modifier l'échelle. Si vous passez une valeur invalide, la propriété est remise à 100% au moment de l'impression.
Option nombre copies	Entier long	4	*valeur1* uniquement : nombre de copies à imprimer
Option alimentation	Entier long	5	(Windows uniqument) *valeur1* uniquement : numéro correspondant à l'indice, dans le tableau des bacs retourné par la commande **VALEURS OPTION IMPRESSION**, du bac papier à utiliser. Cette option est utilisable sous Windows uniquement.
Option couleur	Entier long	8	(Windows uniquement) *valeur1* uniquement : code indiquant le mode de prise en charge de la couleur : 1=Noir et blanc (monochrome), 2=Couleur. **Versions 64 bits** : Cette option n'est pas prise en charge dans les versions 64 bits de 4D (obsolète).
Option destination	Entier long	9	*valeur1*: code indiquant le type de destination de l'impression : 1=Imprimante, 2=Fichier (PC)/PS (Mac), 3=Fichier PDF, 5=Ecran (option du pilote OS X) Si *valeur1* est différent de 1 ou de 5, *valeur2* contient un chemin d'accès pour le document résultant. Ce chemin sera utilisé jusqu'à ce qu'un autre chemin soit spécifié. Si un fichier du même nom existe déjà à l'emplacement de destination, il est remplacé. Avec **LIRE OPTION IMPRESSION**, si la valeur courante n'est pas dans la liste prédéfinie, *valeur1* contient -1 et la variable système OK vaut 1. Si une erreur se produit, *valeur1* et la variable système OK valent 0. **Note :** Sous Windows, vous pouvez définir la destination d'impression 3 (Fichier PDF) lorsque le pilote PDF Creator a été installé. Lorsque les valeurs (*9;3;chemin*) sont passées, 4D lance automatiquement une impression PDF "silencieuse" et prend en compte les codes d'options éventuellement passés (à noter que si vous passez une chaîne vide dans *valeur2* ou omettez ce paramètre, une boîte de dialogue d'enregistrement de fichier apparaît au moment de l'impression). A l'issue de l'impression, les paramétrages courants sont restaurés. Ce principe simplifie le pilotage des impressions PDF par 4D et permet d'écrire du code multi-plate-forme. Si les valeurs (*9;3;chemin*) ne sont pas passées, l'impression n'est pas pilotée par 4D et les éventuels codes d'options de PDF Creator sont ignorés.
Option recto verso	Entier long	11	(Windows uniquement) *valeur1*: 0=Recto ou standard, 1=Recto-verso. Si *valeur1*=1, *valeur2* contient la reliure à appliquer : 0=Reliure à gauche (valeur par défaut), 1=Reliure en haut. **Note :** Cette option est utilisable sous Windows uniquement.
Option nom document à imprimer	Entier long	12	*valeur1* uniquement : nom du document d'impression, qui apparaît dans la liste des documents du serveur d'impression. Le nom défini par cette instruction sera utilisé pour tous les documents d'impression de la session tant qu'un nouveau nom ou une chaîne vide ne sera pas passé(e). Pour utiliser ou rétablir le fonctionnement standard (utilisation du nom de la méthode dans le cas d'une méthode, nom de la table pour un enregistrement, etc.), passez une chaîne vide dans *valeur1*.
Option mode impression Mac	Entier long	13	(Mac uniquement) *valeur1* uniquement : 0=impression en mode PDF (valeur par défaut), 1=impression en mode Postscript. **Notes :** - Cette option n'a pas d'effet sous Windows. - Sous OS X, les impressions sont effectuées par défaut en mode PDF. Or, le pilote d'impression PDF ne prend pas en charge les images PICT encapsulant des informations Postscript — ces images sont générées notamment par des logiciels de dessin vectoriel. Pour résoudre ce problème, cette option permet de modifier le mode d'impression sous OS X pour la session courante. Attention, l'impression en mode Postscript peut entraîner des effets de bords indésirables. **Versions 64 bits :** Cette option n'est pas prise en charge ; elle est remplacée par l'utilisation de l'option Driver PDF générique de la commande **FIXER IMPRIMANTE COURANTE**.
Option masquer progression impr	Entier long	14	*valeur1* uniquement : 1=masquer toutes les fenêtres de progression d'impression, 0=afficher les fenêtres de progression d'impression (fonctionnement par défaut). Cette option est particulièrement utile dans le cadre des impressions en PDF sous OS X. **Note :** Il existe déjà une option d'affichage Progression de l'impression accessible via la boîte de dialogue des Propriétés de la base (page Interface). Toutefois, elle est globale à l'application et ne masque pas toutes les fenêtres sous OS X.
Option intervalle de page	Entier long	15	*valeur1*=numéro de la première page à imprimer (valeur par défaut 1) et (optionnel) *valeur2*=numéro de la dernière page à imprimer (valeur par défaut -1 = fin du document).
Option ancienne couche impression	Entier long	16	(Versions 4D 64 bits pour Windows uniquement) *valeur1* uniquement : 1=sélectionner l'ancienne couche d'impression GDI pour toutes les tâches d'impression suivantes, 0=sélectionner la couche d'impression D2D (défaut). **Versions 64 bits :** Ce sélecteur est pris en charge dans les applications 4D 64 bits monopostes sous Windows uniquement, et est ignoré pour les autres plates-formes. Il est principalement destiné, dans ces applications, à permettre aux plug-ins d'ancienne génération d'imprimer dans des tâches d'impression 4D.

Un code d'option PDF est constitué de deux parties, *TypeOption* et *NomOption*, assemblées sous la forme "*TypeOption:NomOption*". Pour plus d'informations sur les codes d'option PDF et les valeurs possibles, reportez-vous à la description de la commande **FIXER OPTION IMPRESSION**.

Note : La commande **LIRE OPTION IMPRESSION** prend principalement en charge les imprimantes PostScript. Elle peut être utilisée avec d'autres types d'imprimantes, telles que PCL ou lnk, mais dans ce cas il est possible que certaines options ne soient pas disponibles.

Variables et ensembles système

La variable système OK prend la valeur 1 si la commande a été exécutée correctement, sinon elle prend la valeur 0.

Lire taquet impression

Lire taquet impression (numTaquet) -> Résultat

Paramètre	Type		Description
numTaquet	Entier long	→	Numéro de taquet
Résultat	Entier long	←	Position du taquet

Description

La commande **Lire taquet impression** permet de récupérer la position courante d'un taquet lors d'une impression. Les coordonnées sont retournées en pixels (1 pixel = 1/72 pouce).

Cette commande peut être appelée dans deux contextes :

- lors de l'événement formulaire Sur entête, dans le cadre de l'utilisation des commandes **IMPRIMER SELECTION** et **IMPRIMER ENREGISTREMENT**.
- lors de l'événement formulaire Sur impression corps, dans le cadre de l'utilisation de la commande **Imprimer ligne**.

Passez dans le paramètre *numTaquet* une des constantes du thème "**Zone de formulaire**" :

Constante	Type	Valeur
Corps formulaire	Entier long	0
Entête formulaire	Entier long	200
Entête formulaire1	Entier long	201
Entête formulaire10	Entier long	210
Entête formulaire2	Entier long	202
Entête formulaire3	Entier long	203
Entête formulaire4	Entier long	204
Entête formulaire5	Entier long	205
Entête formulaire6	Entier long	206
Entête formulaire7	Entier long	207
Entête formulaire8	Entier long	208
Entête formulaire9	Entier long	209
Pied de page formulaire	Entier long	100
Rupture formulaire0	Entier long	300
Rupture formulaire1	Entier long	301
Rupture formulaire2	Entier long	302
Rupture formulaire3	Entier long	303
Rupture formulaire4	Entier long	304
Rupture formulaire5	Entier long	305
Rupture formulaire6	Entier long	306
Rupture formulaire7	Entier long	307
Rupture formulaire8	Entier long	308
Rupture formulaire9	Entier long	309

Exemple

Reportez-vous à l'exemple de la commande **FIXER TAQUET IMPRESSION**.

LIRE ZONE IMPRESSION

LIRE ZONE IMPRESSION (hauteur {; largeur})

Paramètre	Type		Description
hauteur	Entier long	⇦	Hauteur de la zone d'impression
largeur	Entier long	⇦	Largeur de la zone d'impression

Description

La commande **LIRE ZONE IMPRESSION** retourne dans les paramètres *hauteur* et *largeur* la taille en pixels de la zone d'impression. Cette taille dépend des paramètres d'impression courants, de l'orientation du papier, etc.

Les tailles retournées ne varient pas d'une page à l'autre (après un saut de page par exemple).

Associée à la commande **Lire hauteur imprimee**, cette commande est utile pour connaître le nombre de pixels disponibles pour l'impression, ou pour centrer un objet dans la page.

Pour connaître la taille totale de la page, vous pouvez :

- soit ajouter aux valeurs retournées par cette commande les marges fournies par la commande **LIRE MARGE IMPRESSION**.
- soit utiliser la syntaxe suivante :

```
FIXER MARGE IMPRESSION(0;0;0;0) ` Fixer la marge papier
LIRE ZONE IMPRESSION(hPapier;lPapier) ` Taille du papier
```

Note : Pour plus d'informations sur la gestion des marges d'impression dans 4D, reportez-vous à la description de la commande **LIRE MARGE IMPRESSION**.

LISTE IMPRIMANTES

LISTE IMPRIMANTES (tabNoms {; tabNomsAlt {; tabModèles}})

Paramètre	Type		Description
tabNoms	Tableau texte	⇦	Noms des imprimantes
tabNomsAlt	Tableau texte	⇦	Windows : Emplacements des imprimantes, Mac OS : Noms personnalisés des imprimantes
tabModèles	Tableau texte	⇦	Modèles des imprimantes

Description

La commande **LISTE IMPRIMANTES** remplit le ou les tableau(x) passé(s) en paramètre(s) avec les noms ainsi que, facultativement, les emplacements ou les noms personnalisés et les modèles des imprimantes disponibles pour le poste.

Passez dans le paramètre *tabNoms* le nom d'un tableau texte. Après l'exécution de la commande, ce tableau contiendra la liste des noms d'imprimantes disponibles. Sous Mac OS, il s'agit des noms "système" fixes.

Note : Si les imprimantes sont gérées via un serveur d'impression ("spouleur"), le chemin d'accès complet (sous Windows) ou le nom du spouleur (sous Mac OS) est retourné.

Vous pouvez passer un deuxième tableau facultatif, *tabNomsAlt*. Le contenu de ce tableau dépend de la plate-forme :

- Sous Windows, vous récupérez pour chaque imprimante son emplacement réseau (ou son port local).
- Sous Mac OS, vous récupérez pour chaque imprimante son nom personnalisé, modifiable par l'utilisateur. Ce nom peut être utilisé par exemple dans des boîtes de dialogue.

Le paramètre facultatif *tabModèles* permet de récupérer le modèle de chaque imprimante (**Note :** Ce paramètre n'est pas pris en charge dans les versions Mac 32 bits de 4D).

Utilisez les commandes **FIXER IMPRIMANTE COURANTE** et **Lire imprimante courante** pour modifier ou connaître l'imprimante sélectionnée dans 4D. Vous devez leur passer les noms retournés dans le premier tableau (*tabNoms*).

Sous Windows, le nom d'une imprimante peut être modifié manuellement au niveau du système d'exploitation. En revanche, son emplacement et son modèle sont liés à ses caractéristiques physiques. Vous pouvez donc utiliser les valeurs des tableaux optionnels pour vérifier les caractéristiques de l'imprimante sélectionnée — typiquement, vous pouvez vérifier que tous les clients utilisent la même imprimante.
Sous Mac OS, cette vérification peut s'effectuer sur le nom de l'imprimante (nom du serveur d'impression), qui est le même pour chaque poste connecté.

Variables et ensembles système

La variable système OK prend la valeur 1 si la commande a été exécutée correctement, sinon elle prend la valeur 0 et les tableaux sont retournés vides.

Niveau

Niveau -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Niveau de rupture ou d'en-tête courant

Description

La fonction **Niveau** sert à déterminer le niveau de rupture ou d'en-tête courant. Elle retourne le numéro du niveau de rupture pendant les événements Sur entête et Sur impression sous total.

Le niveau 0 est le dernier niveau à être imprimé et convient à l'impression d'un total général. **Niveau** retourne 1 lorsque 4D imprime une rupture sur le premier champ trié, 2 lorsque 4D imprime une rupture sur le deuxième champ trié, et ainsi de suite.

Exemple

Cet exemple est une maquette de méthode formulaire. Il traite chaque événement possible lorsqu'un état est imprimé dans un formulaire sortie. **Niveau** est appelé lorsqu'un en-tête ou une rupture est imprimé(e) :

```
   ` Méthode formulaire pour un formulaire sortie utilisé pour un état
$vpFormTable:=Table du formulaire courant
Au cas ou
   ` ...
   :(Evenement formulaire=Sur entête)
   ` La zone en-tête va être imprimée
      Au cas ou
         :(Avant selection($vpFormTable->))
   ` Le code pour la première rupture d'en-tête doit être placé ici
         :(Niveau=1)
   ` Le code pour la rupture d'en-tête niveau 1 doit être placé ici
         :(Niveau=2)
   ` Le code pour la rupture d'en-tête niveau 2 doit être placé ici
   ` ...
      Fin de cas
   :(Evenement formulaire=Sur impression corps)
   ` Un enregistrement va être imprimé
   ` Le code pour chaque enregistrement doit être placé ici
   :(Evenement formulaire=Sur impression sous total)
   ` Une rupture va être imprimée
      Au cas ou
         :(Niveau=0)
   ` Le code pour la rupture 0 doit être placé ici
         :(Niveau=1)
   ` Le code pour la rupture 1 doit être placé ici
   ` ...
      Fin de cas
   :(Evenement formulaire=Sur impression pied de page)
      Si(Fin de selection($vpFormTable->))
   ` Le code pour le dernier pied de page doit être placé ici
      Sinon
   ` Le code pour le pied de page doit être placé ici
      Fin de si
Fin de cas
```

NIVEAUX DE RUPTURES

NIVEAUX DE RUPTURES (niveau {; sautPage})

Paramètre	Type		Description
niveau	Entier long	⇨	Nombre de niveaux de rupture
sautPage	Entier long	⇨	Niveau de saut de page

Description

NIVEAUX DE RUPTURES spécifie le nombre de niveaux de rupture dans un état créé à l'aide de la commande **IMPRIMER SELECTION**.

Vous **devez** appeler **NIVEAUX DE RUPTURES** et **CUMULER SUR** avant la génération de chaque état dans lequel vous voulez utiliser des ruptures. Ces commandes activent le traitement des ruptures pour un état. Pour plus d'informations, reportez-vous à la description de la commande **Sous total**.

Le paramètre *niveau* indique le dernier niveau de rupture pour lequel vous voulez utiliser des ruptures. Ce nombre doit être inférieur ou égal aux niveaux de tris que vous aurez effectués avant l'impression. Si vous avez effectué un tri sur davantage de niveaux, ces niveaux seront imprimés triés, mais ne comporteront pas de rupture.

Chaque niveau de rupture généré provoquera l'impression de zones de rupture et d'en-tête dans le formulaire. Il doit y avoir au moins autant de zones de rupture dans le formulaire que la valeur que vous avez passée dans *niveau*. S'il y a davantage de zones de rupture, elles seront ignorées et ne seront pas imprimées.

Le second paramètre (optionnel), *sautPage*, permet de provoquer un saut de page sur le niveau de rupture de votre choix.

Exemple

L'exemple suivant imprime un état avec deux niveaux de rupture. La sélection est triée sur quatre champs, mais la commande **NIVEAUX DE RUPTURES** ne spécifie que deux niveaux de rupture. Seul un champ est cumulé à l'aide de la commande **CUMULER SUR** :

```
TRIER([Emp]Service;>;[Emp]Titre;>;[Emp]Nom;>;Emp]Prénom;>) ` Trier sur quatre champs
NIVEAUX DE RUPTURES(2) ` Fixer 2 niveaux de rupture (Service et Titre)
CUMULER SUR([Emp]Salaire) ` Cumuler sur les salaires
FORM FIXER SORTIE([Emp];"ServiceRessHum") ` Sélectionner le formulaire à imprimer
IMPRIMER SELECTION([Emp]) ` Imprimer l'état
```

OUVRIR TACHE IMPRESSION

OUVRIR TACHE IMPRESSION
Ne requiert pas de paramètre

Description

La commande **OUVRIR TACHE IMPRESSION** ouvre une tâche d'impression (print job) et y empile tous les ordres d'impression exécutés par la suite, tant que la commande **FERMER TACHE IMPRESSION** n'est pas appelée. Cette commande vous permet de contrôler les tâches d'impression, et notamment de vous assurer qu'aucune tâche d'impression "parasite" ne puisse s'intercaler dans une séquence d'impressions.

La commande **OUVRIR TACHE IMPRESSION** peut être utilisée avec toutes les commandes d'impression de 4D, les commandes de l'éditeur d'états rapides ainsi que les commandes d'impression des plug-ins 4D Write et 4D View. En revanche, cette commande n'est pas compatible avec les plug-ins 4D Chart et 4D Draw, ainsi que la plupart des plug-ins tiers.

Lorsqu'une tâche est ouverte avec cette commande, l'imprimante est placée en mode "occupé" jusqu'à ce que l'impression soit effectivement lancée. Si un plug-in non compatible lance une impression dans ce contexte, l'erreur "imprimante occupée" est retournée.

Vous devez appeler la commande **FERMER TACHE IMPRESSION** pour terminer la tâche et envoyer le document d'impression à l'imprimante. Si vous omettez cette commande, le document d'impression restera dans la pile et l'imprimante sera inaccessible jusqu'à ce que vous quittiez l'application 4D.

La tâche d'impression est locale au process mais l'impression s'effectuant au niveau global, une seule tâche d'impression peut être ouverte à la fois dans 4D, tous process confondus.

OUVRIR TACHE IMPRESSION utilise les paramètres d'impression courants (paramètres par défaut ou définis via les commandes **UTILISER PARAMETRES IMPRESSION** et/ou **FIXER OPTION IMPRESSION**). Les commandes modifiant les paramètres d'impression doivent être appelées avant **OUVRIR TACHE IMPRESSION**. Dans le cas contraire, une erreur est générée.

Page impression

Page impression -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Numéro de la page en cours d'impression

Description

Page impression retourne le numéro de la page en cours d'impression. Cette fonction vous permet de numéroter automatiquement les pages d'une impression en cours à l'aide de **IMPRIMER SELECTION** ou du menu Impression dans le mode Développement.

Exemple

L'exemple suivant change la position des numéros de page sur un état pour que l'état puisse être reproduit au format recto-verso. Le formulaire pour l'état comporte deux variables qui affichent les numéros de page. Une variable dans le coin bas à gauche (*vNumGauche*) imprime les numéros de page pairs. Une autre variable dans le coin bas à droite (*vNumDroite*) imprime les numéros de page impairs. L'exemple teste si le numéro de page est pair ou impair, puis utilise et efface les variables appropriées :

```
Au cas ou
   :(Evenement formulaire=Sur impression pied de page)
      Si((Page impression% 2)=0) ` Modulo vaut 0 pour un numéro de page pair
         vNumGauche:=Chaine(Page impression) ` Fixer le numéro de page à gauche
         vNumDroite:="" ` Effacer le numéro de page droit
      Sinon ` Sinon, le numéro de page est impair
         vNumGauche:="" ` Effacer le numéro de page gauche
         vNumDroite:=Chaine(Page impression) ` Fixer le numéro de page à droite
      Fin de si
Fin de cas
```

PARAMETRES IMPRESSION

PARAMETRES IMPRESSION {(typeDial)}

Paramètre	Type		Description
typeDial	Entier long	⇨	Boîte(s) de dialogue à afficher

Description

La commande **PARAMETRES IMPRESSION** provoque l'affichage d'une ou des deux boîtes de dialogue de paramétrage d'impression. Cette commande doit être appelée avant une série de commandes **Imprimer ligne** ou la commande **OUVRIR TACHE IMPRESSION**.

Le paramètre facultatif *typeDial* permet de configurer l'affichage des boîtes de dialogue d'impression. Vous pouvez utiliser l'une des constantes suivantes du thème **Options d'impression**. La ou les boîte(s) de dialogue affichée(s) dépend de votre version de 4D, comme le montre le tableau suivant :

typeDial (constante)	4D 64 bits	4D 32 bits
0 ou omis	Impression	Format d'impression+Impression
1 (Dialogue de format d'impression)	Format d'impression	Format d'impression
2 (Dialogue d'impression)	Impression (= 0 ou omis)	Impression

Note : La boîte de dialogue d'impression comporte l'option **Imprimer à l'écran** permettant à l'utilisateur de visualiser son impression à l'écran. Vous pouvez présélectionner ou désélectionner cette option par un appel préalable à la commande **FIXER APERCU IMPRESSION**.

Exemple

Reportez-vous à l'exemple de la commande **Imprimer ligne**.

Variables et ensembles système

Si l'utilisateur clique sur le bouton OK dans chaque boîte de dialogue, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

Paramètres impression vers BLOB

Paramètres impression vers BLOB (paramImpression) -> Résultat

Paramètre	Type		Description
paramImpression	BLOB	⇐	Paramètres courants d'impression
Résultat	Entier long	⇒	Code d'état : 1=Opération réussie, 0=Pas d'imprimante courante

Description

La commande **Paramètres impression vers BLOB** sauvegarde les paramètres d'impression courants de 4D dans le BLOB *paramImpression*. Le paramètre *paramImpression* stocke tous les paramètres utilisés pour l'impression :

- Paramètres de mise en page comme le papier, l'orientation, l'échelle...
- Paramètres d'impression comme le nombre de copies, la source du papier...

Cette commande doit être utilisée conjointement avec la commande **BLOB vers paramètres impression**. Ces commandes vous permettent de sauvegarder les paramètres d'impression de l'utilisateur courant et de les recharger pour qu'il n'ait pas à préciser ses paramètres chaque fois qu'il imprime. De plus, cela permet de garder des paramètres d'impression "privés" (spécifiques à un pilote d'imprimante) qui ne sont pas disponibles dans les paramètres d'impression standard.

Le BLOB généré ne doit pas être modifié par programmation : il ne peut être utilisé qu'avec la commande **BLOB vers paramètres impression**.

La commande retourne 1 si le BLOB a été correctement généré et 0 si aucune imprimante courante n'est sélectionnée.

Windows / OS X

Le BLOB *paramImpression* peut être sauvegardé et lu sur les deux plateformes. Toutefois, même si certains paramètres d'impression sont communs, d'autres sont spécifiques à la plateforme et dépendent du pilote d'impression et des versions de l'OS. Si le même BLOB *paramImpression* est partagé entre les deux plateformes, vous pouvez perdre des informations.

Lorsque vous utilisez un environnement hétérogène, pour restaurer le maximum de paramètres d'impression disponibles pour chaque plateforme (et pas seulement la partie commune), il est recommandé de gérer deux BLOBs *paramImpression*, un pour chaque plateforme.

Exemple

Vous voulez sauvegarder les paramètres d'impression courants sur disque :

```
C_BLOB(curSettings)
PARAMETRES IMPRESSION //displays print settings dialog to the user
Si(OK=1)
   $err:=Paramètres impression vers BLOB(curSettings)
   Si($err=1)
      BLOB VERS DOCUMENT(Dossier 4D+"current4Dsettings.blob";curSettings)
   Sinon
      ALERTE("Pas d'imprimante sélectionnée")
   Fin de si
Fin de si
```

SAUT DE PAGE

SAUT DE PAGE {(* | >)}

Paramètre	Type		Description
* | >		⇨	* Annule l'impression lancée par Imprimer ligne ou > Rend l'impression prioritaire

Description

La commande **SAUT DE PAGE** déclenche l'impression des données envoyées à l'imprimante et provoque un saut de page. **SAUT DE PAGE** s'utilise conjointement avec **Imprimer ligne** (dans le cadre de l'événement formulaire Sur impression corps) pour forcer des sauts de page et imprimer la dernière page créée en mémoire.
N'appelez pas **SAUT DE PAGE** avec la commande **IMPRIMER SELECTION** : dans ce cas, il est préférable d'utiliser les routines **Sous total** ou **NIVEAUX DE RUPTURES** avec leur paramètre optionnel pour générer des sauts de pages.

Les paramètres * et > sont optionnels.

Le paramètre * vous permet d'annuler une impression lancée avec la commande **Imprimer ligne**. L'exécution de cette instruction stoppe immédiatement l'impression en cours.

Note : Sous Windows, ce mécanisme peut être perturbé par les propriétés de "spouling" du serveur d'impression. Si l'imprimante est paramétrée de manière à lancer les impressions directement, l'annulation ne sera pas effective. Pour que l'instruction **SAUT DE PAGE(*)** fonctionne correctement, il est préférable d'affecter la propriété "Commencer l'impression une fois la dernière page spoulée" à l'imprimante.

Le paramètre > modifie le fonctionnement de la commande **SAUT DE PAGE**. Cette syntaxe provoque deux effets :

- Elle reporte l'impression jusqu'à ce que l'instruction **SAUT DE PAGE** sans paramètre soit de nouveau exécutée.
- Elle rend l'impression prioritaire. Aucune autre impression ne pourra s'intercaler tant que celle en cours ne sera pas terminée.
  Cette seconde option est particulièrement intéressante lorsqu'elle est utilisée dans le cadre d'impressions en files d'attente. Le paramètre > garantit que l'impression sera réalisée à partir d'un seul fichier. Cela permet de réduire le temps d'impression.

Note : Lors d'une impression avec aperçu, si l'utilisateur clique sur le bouton d'annulation dans la boîte de dialogue de prévisualisation, la commande **SAUT DE PAGE** met la variable système OK à 0.

Exemple 1

Reportez-vous à l'exemple de la commande **Imprimer ligne**.

Exemple 2

Reportez-vous à l'exemple de la commande **FIXER TAQUET IMPRESSION**.

Sous total

Sous total (valeurs {; sautPage}) -> Résultat

Paramètre	Type		Description
valeurs	Champ	➡	Champ ou variable numérique dont vous voulez calculer le sous-total
sautPage	Entier long	➡	Niveau de rupture auquel effectuer un saut de page
Résultat	Réel	⮌	Sous-total de valeurs

Description

Sous total retourne le sous-total de *valeurs* pour le niveau de rupture courant ou précédent. **Sous total** ne fonctionne que dans le cadre d'une sélection triée imprimée par l'intermédiaire de la commande **IMPRIMER SELECTION** ou de la commande de menu **Imprimer** du mode Développement. Le paramètre *valeurs* doit être de type numérique, entier ou entier long. Vous devez assigner le résultat de la fonction **Sous total** à une variable placée dans la zone de rupture du formulaire.

Attention : Vous **devez** utiliser les commandes **NIVEAUX DE RUPTURES** et **CUMULER SUR** avant d'imprimer un état sur lequel vous voulez traiter les niveaux de rupture et calculer des sous-totaux. Reportez-vous au paragraphe situé à la fin de cette section.

Le second paramètre (optionnel) de la fonction **Sous total** est utilisé pour provoquer des sauts de page lors de l'impression. Si *sautPage* vaut 0, **Sous total** ne génère aucun saut de page. Si *sautPage* vaut 1, **Sous total** génère un saut de page pour chaque niveau de rupture 1. Si *sautPage* vaut 2, **Sous total** génère un saut de page pour chaque niveau de rupture 1 et 2, etc.

Conseil : Si vous faites appel à la fonction **Sous total** dans le formulaire sortie affiché à l'écran, 4D va afficher un message d'erreur. La fermeture du dialogue d'erreur va provoquer un rafraîchissement de l'écran, donc de nouveau l'exécution de la méthode qui fait appel à **Sous total**, donc de nouveau un message d'erreur, etc. Pour sortir de ce cercle vicieux, appuyez sur les touche **Alt** + **Maj** (Windows) ou **Option**+**Maj** (Macintosh) et cliquez sur le bouton **Arrêter** dans la fenêtre d'erreur : cela met provisoirement fin aux rafraîchissements d'écran. Choisissez un autre formulaire de sortie pour éviter que le problème ne se répète. Passez en mode Structure pour isoler l'appel à la fonction **Sous total** par un test (**Evenement formulaire** = Sur impression sous total) si vous avez l'intention d'utiliser le même formulaire de sortie pour l'écran et l'imprimante.

Exemple

L'exemple suivant est la méthode objet d'une variable intitulée *vSalaire*, placée dans une zone de rupture d'un formulaire (R0, la zone située au-dessus du marqueur R0). La variable prend la valeur du sous-total du champ Salaire pour ce niveau de rupture. Le traitement des ruptures doit avoir été auparavant activé par les commandes **NIVEAUX DE RUPTURES** et **CUMULER SUR**.

```
Au cas ou
   :(Evenement formulaire=Sur impression sous total)
      vSalaire:=Sous total([Employés]Salaire)
Fin de cas
```

Reportez-vous au chapitre "Les formulaires de sortie et les états" du manuel Mode Développement pour plus d'informations sur la construction de formulaires avec des niveaux de ruptures.

Traitement de niveaux de rupture dans les formulaires d'état

Pour pouvoir générer des états avec ruptures, vous devez déclencher le traitement des ruptures en appelant les commandes **NIVEAUX DE RUPTURES** et **CUMULER SUR**. Il faut que ces deux commandes soient appelées avant l'impression du formulaire. L'appel à la fonction **Sous total** est nécessaire pour afficher les calculs de niveaux intermédiaires. Il est obligatoire de trier sur au moins le nombre de niveaux de ruptures désiré.

Dans le cadre de l'utilisation des commandes **NIVEAUX DE RUPTURES** et **CUMULER SUR**, les étapes à suivre sont :
1. Sélectionner les enregistrements à imprimer,
2. Trier les enregistrements sur autant de niveaux que de niveaux de ruptures,
3. Appeler les commandes **NIVEAUX DE RUPTURES** et **CUMULER SUR**,
4. Imprimer l'état avec la commande **IMPRIMER SELECTION**.

La commande **Sous total** permet d'afficher des calculs de sous-totaux dans des formulaires.

UTILISER PARAMETRES IMPRESSION

UTILISER PARAMETRES IMPRESSION ({laTable ;} formulaire)

Paramètre	Type		Description
laTable	Table	⇒	Table à laquelle appartient le formulaire ou Table par défaut si ce paramètre est omis
formulaire	Chaîne	⇒	Formulaire à utiliser pour définir les paramètres d'impression

Description

UTILISER PARAMETRES IMPRESSION spécifie qu'une impression doit utiliser les paramètres mémorisés avec *formulaire*. Les paramètres d'impression sont stockés avec le formulaire au moment où il est sauvegardé en mode Développement.

Dans les trois cas suivants :

- un appel à **IMPRIMER SELECTION** auquel vous passez le paramètre optionnel *,
- un appel à **IMPRIMER ENREGISTREMENT** auquel vous passez le paramètre optionnel *,
- une série d'appels à **Imprimer ligne** non précédée d'un appel à **PARAMETRES IMPRESSION**,

... les boîtes de dialogue d'impression ne sont pas affichées et l'impression est effectuée avec les paramètres par défaut. Appeler **UTILISER PARAMETRES IMPRESSION** vous permet dans ce cas de ne pas afficher les boîtes de dialogue d'impression ET d'utiliser des paramètres d'impression qui ne sont pas ceux par défaut.

Exemple

Plusieurs formulaires (vides) sont créés pour une table nommée [Dessins]. Le format de page du formulaire “PS100” est fixé à 100%, le format de page du formulaire “PS90” est fixé à 90%, etc. La méthode projet suivante vous permet d'imprimer la sélection d'une table à différentes échelles sans devoir à chaque fois spécifier manuellement l'échelle dans les boîtes de dialogue d'impression (qui ne sont d'ailleurs pas affichées) :

```
   ` Méthode projet IMPRESSION ECHELLE AUTO
   ` IMPRESSION ECHELLE AUTO ( Pointeur ; Chaine {; Entier long } )
   ` IMPRESSION ECHELLE AUTO ( ->[Table]; "Form Sortie" {; Echelle } )
Si(Nombre de paramètres>=3)
   UTILISER PARAMETRES IMPRESSION([Dessins];"PS"+Chaine($3))
   Si(Nombre de paramètres>=2)
      FORMULAIRE SORTIE($1->;$2)
   Fin de si
Fin de si
Si(Nombre de paramètres>=1)
   IMPRIMER SELECTION($1->;*)
Sinon
   IMPRIMER SELECTION(*)
Fin de si
```

Une fois que cette méthode projet est écrite, vous pouvez l'appeler ainsi :

```
   ` Recherche des factures courantes
CHERCHER([Factures];[Factures]Payé=Faux)
   ` Impression d'un état réduit à 90%
IMPRESSION ECHELLE AUTO(->[Factures];"Etat Résumé";90)
   ` Impression d'un état réduit à 50%
IMPRESSION ECHELLE AUTO(->[Factures];"Etat détaillé";50)
```

VALEURS OPTION IMPRESSION

VALEURS OPTION IMPRESSION (option ; tabNoms {; tabInfo1 {; tabInfo2}})

Paramètre	Type		Description
option	Entier long	➡	Numéro d'option
tabNoms	Tableau texte	⬅	Noms des valeurs
tabInfo1	Tableau entier long	⬅	Valeurs 1 de l'option
tabInfo2	Tableau entier long	⬅	Valeurs 2 de l'option

Description

La commande **VALEURS OPTION IMPRESSION** retourne dans le tableau *tabNoms* la liste des noms de valeurs disponibles pour l'*option* d'impression définie. Facultativement, vous pouvez récupérer des informations sur chaque valeur dans les tableaux *tabInfo1* et *tabInfo2*.

Le paramètre *option* vous permet de désigner l'option à lire. Vous devez passer une des constantes du thème **Options d'impression** suivantes (options pouvant retourner des listes de noms de valeurs) :

Constante	Type	Valeur	Comment
Option alimentation	Entier long	5	(Windows uniqument) *valeur1* uniquement : numéro correspondant à l'indice, dans le tableau des bacs retourné par la commande **VALEURS OPTION IMPRESSION**, du bac papier à utiliser. Cette option est utilisable sous Windows uniquement.
Option papier	Entier long	1	Si vous passez uniquement *valeur1*, il contient le nom du papier. Si vous passez les deux paramètres, *valeur1* contient la largeur du papier et *valeur1* contient la hauteur du papier. La largeur et la hauteur sont exprimées en pixels écran. Utilisez la commande **VALEURS OPTION IMPRESSION** pour connaître le nom, la hauteur et la largeur de tous les formats de papier proposés par l'imprimante.

Après exécution de la commande, le tableau *tabNoms* ainsi que, le cas échéant, les tableaux *tabInfo1* et *tabInfo2* seront remplis par la commande avec les noms et informations des valeurs disponibles.

Si vous passez la valeur 1 (Option papier) dans le paramètre *option*, la commande retournera les informations suivantes :

- dans le tableau *tabNoms*, les noms des formats de papiers disponibles ;
- dans le tableau *tabInfo1*, les hauteurs de chaque format de papier ;
- dans le tableau *tabInfo2*, les largeurs de chaque format de papier.

Note : Pour que vous puissiez obtenir ces informations, le pilote d'imprimante doit avoir accès à un fichier de description PostScript (PPD) valide de l'imprimante.

Pour utiliser un format de papier spécifique à l'aide de la commande **FIXER OPTION IMPRESSION**, vous pouvez passer soit une des valeurs du tableau *tabNoms*, soit les valeurs correspondantes des tableaux *tabInfo1* et *tabInfo2*.

Si vous passez la valeur 5 (Option alimentation) dans le paramètre *option*, la commande retourne dans le tableau *tabNoms* les noms des différents bacs disponibles et leur numéro Windows interne dans *tabInfo1* (*tabInfo2* reste vide).
L'ordre des valeurs dans les tableaux est défini par le pilote d'impression. Pour désigner un bac à l'aide de la commande **FIXER OPTION IMPRESSION**, vous devez passer l'indice de l'élément souhaité dans tableau *tabNoms* ou *tabInfo1*.

Note : Cette option est utilisable sous Windows uniquement.

Pour plus d'informations sur les différentes options d'impression, reportez-vous à la description des commandes **FIXER OPTION IMPRESSION** et **LIRE OPTION IMPRESSION**.

Toutes les informations retournées par ces commandes sont fournies par le système d'exploitation. Reportez-vous à la documentation de votre système pour plus de détails sur certaines options.

Note : La commande **VALEURS OPTION IMPRESSION** fonctionne avec les imprimantes PostScript uniquement.

Interface utilisateur

- AFFICHER BARRE MENUS
- APPELER SUR A PROPOS
- BEEP
- CACHER BARRE MENUS
- CHANGER POINTEUR SOURIS
- FIXER TITRES CHAMPS
- FIXER TITRES TABLES
- GENERER CLIC
- GENERER EVENEMENT
- GENERER FRAPPE CLAVIER
- JOUER SON
- LIRE TITRES CHAMPS
- LIRE TITRES TABLES
- Macintosh commande enfoncee
- Macintosh control enfoncee
- Macintosh option enfoncee
- Majuscule enfoncee
- Objet focus
- Pop up menu
- POSITION SOURIS
- REDESSINER
- Verrouillage majuscule enfoncee
- Windows Alt enfoncee
- Windows Ctrl enfoncee
- *_o_FIXER INTERFACE*
- *_o_INVERSER FOND*
- *_o_Lire interface*

AFFICHER BARRE MENUS

AFFICHER BARRE MENUS
Ne requiert pas de paramètre

Description

La commande **AFFICHER BARRE MENUS** rend visible la barre de menus.
Si la barre de menus était déjà visible, cette commande ne fait rien.

Exemple

Reportez-vous à l'exemple de la commande **CACHER BARRE MENUS**.

APPELER SUR A PROPOS

APPELER SUR A PROPOS (libelléElément ; méthode)

Paramètre	Type		Description
libelléElément	Chaîne	⇒	Nouvelle ligne de menu A propos...
méthode	Chaîne	⇒	Nom de la méthode à exécuter lorsque la ligne est choisie

Description

La commande **APPELER SUR A PROPOS** remplace la ligne de menu **A propos de 4D...** du menu **Aide** (sous Windows) ou du menu **application** (Mac OS X) par *libelléLigne*.

Après l'appel de cette commande, lorsqu'un utilisateur sélectionne la ligne de menu en mode Développement ou Application, la méthode *méthode* est appelée. Typiquement, cette méthode affiche une boîte de dialogue qui fournit des informations sur les versions de votre base.

Cette commande est utilisable avec 4D (tous modes), 4D Desktop et 4D Server. Son exécution sur le poste serveur provoque la création d'un nouveau process.

Exemple 1

L'exemple suivant remplace la commande de menu **A propos** par la commande de menu **A propos du programmateur**. La méthode ***A PROPOS*** affiche une fenêtre d'A propos personnalisée :

```
APPELER SUR A PROPOS("A propos du programmateur";"A PROPOS")
```

Exemple 2

L'exemple suivant réinitialise la commande de menu d'A propos de 4D :

```
APPELER SUR A PROPOS("A propos de 4D®";"")
```

BEEP

BEEP
Ne requiert pas de paramètre

Description

La commande **BEEP** provoque l'émission d'un bip sonore. Votre PC ou votre Macintosh peut émettre un autre son qu'un bip en fonction du son sélectionné dans le tableau de bord de contrôle du son.

ATTENTION : N'appelez pas la commande **BEEP** à partir d'un process de connexion Web car le bip sonore se produira sur le poste serveur Web 4D et non sur le poste du navigateur Web.

Exemple

Dans l'exemple suivant, un bip est émis et une alerte affichée lorsqu'aucun enregistrement n'est trouvé par une recherche :

```
CHERCHER([Clients];[Clients]Nom=$vsNomAChercher)
Si(Enregistrements trouves([Clients])=0)
   BEEP
   ALERTE("Il n'y a aucun client de ce nom.")
Fin de si
```

CACHER BARRE MENUS

CACHER BARRE MENUS
Ne requiert pas de paramètre

Description

La commande **CACHER BARRE MENUS** rend invisible la barre de menus.
Si la barre de menus était déjà cachée, la commande est sans effet.

Exemple

La méthode suivante passe un enregistrement en plein écran (sous Mac OS) jusqu'à ce que l'utilisateur clique sur le bouton de la souris :

```
CACHER BARRE OUTILS
CACHER BARRE DE MENUS
Creer fenetre(-1;-1;1+Largeur ecran;1+Hauteur ecran;Dialogue modal)
FORM FIXER ENTREE([Tableaux];"Plein écran 800")
AFFICHER ENREGISTREMENT([Tableaux])
Repeter
   POSITION SOURIS($vlX;$vlY;$vlBouton)
Jusque($vlBouton#0)
FERMER FENETRE
AFFICHER BARRE MENUS
AFFICHER BARRE OUTILS
```

Note : Sous Windows, la taille de la fenêtre sera limitée par celle de la fenêtre de l'application.

CHANGER POINTEUR SOURIS

CHANGER POINTEUR SOURIS {(curseur)}

Paramètre	Type		Description
curseur	Entier long	⇨	Numéro de curseur système

Description

La commande **CHANGER POINTEUR SOURIS** remplace le pointeur (graphique) de la souris par le pointeur système dont vous avez passé le numéro d'ID dans le paramètre *curseur*.

Cette commande doit être appelée dans le contexte de l'**Evenement formulaire** Sur survol.

Pour restaurer le pointeur de souris standard, appelez la commande sans paramètre.

Voici les curseurs disponibles :

Numéro	Curseur
1	
2	
4	
9000	
9001	
9003	
9004	
9005	
9006	
9021	
351	
9010	
9011	
9013	
9014	
9015	
9016	
9017	
9019	
9020	
559	
560	

Note : La disponibilité et l'apparence des curseurs varie en fonction du système d'exploitation.

Exemple

Vous souhaitez que le curseur se transforme en  lorsque la souris survole une variable du formulaire. Dans la méthode de la variable, vous pouvez écrire :

```
Si(Evenement formulaire=Sur survol)
   CHANGER POINTEUR SOURIS(9019)
Fin de si
```

FIXER TITRES CHAMPS

FIXER TITRES CHAMPS (laTable ; titresChamps ; numChamps {; *})

Paramètre	Type		Description
laTable	Table	⇒	Table dont vous voulez redéfinir les titres des champs
titresChamps	Tableau chaîne	⇒	Nouveaux titres des champs
numChamps	Tableau entier long	⇒	Numéros des champs
*		⇒	Utiliser les noms personnalisés dans l'éditeur de formules

Description

FIXER TITRES CHAMPS vous permet de masquer, renommer et réordonner les champs d'une table de votre base lorsqu'ils apparaissent dans les éditeurs standard de 4D en mode Application (lorsque les éditeurs sont appelés via des commandes du langage de 4D). Par exemple, cette commande peut modifier l'affichage des tables dans l'éditeur de recherches en mode Application.
Cette commande vous permet également de modifier les libellés des champs apparaissant dans vos formulaires, si vous avez utilisé des libellés dynamiques. Pour plus d'informations sur l'insertion de libellés de tables et de champs dynamiques dans les formulaires, reportez-vous au manuel Mode Développement.

Les tableaux *titresChamps* et *numChamps* doivent être synchronisés. Dans le tableau *titresChamps*, vous passez les noms des champs tels que vous voulez qu'ils apparaissent. Les champs s'afficheront dans l'ordre défini par ces tableaux. Dans chaque élément du tableau *numChamps*, vous passez le numéro de la table qui correspond au nom, nouveau ou ancien, du champ, et ce dans le même numéro d'élément que dans le tableau *titresChamps*.
Si vous voulez masquer un champ, il suffit de ne pas le passer dans les tableaux. Vous avez, par exemple, une table comprenant les champs F, G et H, créés dans cet ordre. Vous voulez que ces champs apparaissent comme M, N et O. De plus, vous ne voulez pas faire apparaître le champ N. Enfin, vous voulez que les tables soient dans l'ordre O et M. Pour cela, vous passez dans le paramètre *titresChamps* un tableau contenant deux éléments, O et M, et vous passez dans le paramètre *numChamps* un tableau contenant deux éléments, 3 et 1.

Le paramètre facultatif * vous permet d'indiquer si les noms personnalisés définis à l'aide de cette commande peuvent être ou non utilisés dans les formules de 4D.

- Par défaut, lorsque ce paramètre est omis, les formules exécutées dans 4D ne peuvent pas utiliser ces noms personnalisés ; il est nécessaire d'utiliser les noms de champs véritables. Ce principe permet une certaine liberté dans le nommage des champs, car l'interpréteur du langage ne traite pas les noms personnalisés.
- Si le paramètre * est passé, les noms définis par cette commande peuvent être utilisés dans les formules exécutées par 4D. **Attention dans ce cas**, les noms personnalisés ne doivent pas contenir de caractères “interdits” par l’interpréteur du langage de 4D, tels que -?*! (pour plus d'informations, reportez-vous à la section “**Nommer les objets du langage 4D**”).

Note : Si votre application donne accès à l'éditeur de formules (par exemple via l'éditeur d'états rapides), il est nécessaire de passer le paramètre * afin d'assurer la cohérence de l'interface.

FIXER TITRES CHAMPS ne modifie pas la structure de votre base. Elle n'affecte que l'affichage ultérieur des éditeurs standard de 4D et des formulaires comportant des libellés dynamiques lorsqu'ils sont appelés via une commande de langage (la structure réelle de la base est affichée lorsque l'éditeur ou le formulaire est appelé depuis une commande de menu en mode Développement). L'aire de validité de la commande **FIXER TITRES CHAMPS** est la session. L'avantage, en Client/Serveur, est que plusieurs 4D distants peuvent “voir” simultanément votre structure d'une manière différente. Vous pouvez appeler **FIXER TITRES CHAMPS** autant de fois que vous voulez.

La commande **FIXER TITRES CHAMPS** est utile dans les cas suivants :

- Traduction dynamique de votre base.
- Affichage des champs dans l'ordre et avec les noms que vous voulez, indépendamment de leurs définitions.
- Affichage des champs suivant l'identité ou les privilèges d'un utilisateur.

ATTENTION :

- **FIXER TITRES CHAMPS** n'annule pas l'effet de la propriété Invisible d'un champ. Si vous avez défini un champ en tant qu'invisible au niveau de la structure, il n'apparaîtra pas en mode Application même s'il est spécifié dans **FIXER TITRES CHAMPS**.
- Les plug-ins accèdent toujours à la structure "virtuelle" telle que définie par cette commande.

Exemple

Reportez-vous à l'exemple de la commande **FIXER TITRES TABLES**.

FIXER TITRES TABLES

FIXER TITRES TABLES (titresTables ; numTables {; *})

Paramètre	Type		Description
titresTables	Tableau chaîne	⇒	Noms des tables tels qu'ils doivent apparaître
numTables	Tableau entier long	⇒	Numéros des tables
*		⇒	Utiliser les noms personnalisés dans l'éditeur de formules

Description

FIXER TITRES TABLES vous permet de masquer, renommer et réordonner les tables de votre base qui apparaissent dans les éditeurs standard de 4D en mode Application (lorsque ces éditeurs sont appelés via des commandes du langage de 4D). Par exemple, cette commande peut modifier l'affichage des tables dans l'éditeur de recherches en mode Application.
Cette commande vous permet également de modifier les libellés des tables apparaissant dans vos formulaires, si vous avez utilisé des libellés dynamiques. Pour plus d'informations sur l'insertion de libellés de tables et de champs dynamiques dans les formulaires, reportez-vous à la section **Utiliser des références dans les textes statiques** dans le manuel *Mode Développement*.

Les tableaux *titresTables* et *numTables* doivent être synchronisés. Dans le tableau *titresTables*, vous passez les noms des tables tels que vous voulez qu'ils apparaissent. Les tables s'afficheront dans l'ordre défini par ces tableaux. Dans chaque élément du tableau *numTables*, passez le numéro de la table qui correspond au nom, nouveau ou ancien, de la table, et ce dans le même numéro d'élément que dans le tableau *titresTables*.
Si vous voulez masquer une table, ne la mettez pas dans les tableaux. Vous avez, par exemple, une base comprenant les tables A, B et C, créées dans cet ordre. Vous voulez que ces tables apparaissent sous les noms X, Y et Z. De plus, vous ne voulez pas faire apparaître la table B. Enfin, vous voulez que les tables soient dans l'ordre Z et X. Pour cela, vous passez dans le paramètre *titresTables* un tableau à deux éléments, Z et X, et vous passez dans le paramètre *numTables* un tableau à deux éléments, 3 et 1.

Le paramètre facultatif * vous permet d'indiquer si les noms personnalisés définis à l'aide de cette commande peuvent être ou non utilisés dans les formules de 4D :

- Par défaut, lorsque ce paramètre est omis, les formules exécutées dans 4D ne peuvent pas utiliser ces noms personnalisés ; il est nécessaire d'utiliser les noms de tables véritables. Ce principe permet une certaine liberté dans le nommage des tables, car l'interpréteur du langage ne traite pas les noms personnalisés.
- Si le paramètre * est passé, les noms définis par cette commande peuvent être utilisés dans les formules exécutées par 4D. **Attention dans ce cas**, les noms personnalisés ne doivent pas contenir de caractères “interdits” par l’interpréteur du langage de 4D, tels que -?*%! Par exemple, le nom "Taux_en_%" ne pourra pas être utilisé dans une formule (pour plus d'informations, reportez-vous à la section “**Nommer les objets du langage 4D**”).

Note : Si votre application donne accès à l'éditeur de formules (par exemple via l'éditeur d'états rapides), il est nécessaire de passer le paramètre * afin d'assurer la cohérence de l'interface.

FIXER TITRES TABLES ne modifie pas la structure de votre base. Cette commande n'affecte que l'utilisation ultérieure des éditeurs standard de 4D et des formulaires comportant des libellés dynamiques lorsqu'ils sont appelés via une commande de langage (la structure réelle de la base est affichée lorsque l'éditeur ou le formulaire est appelé depuis une commande de menu en mode Développement). L'aire de validité de la commande **FIXER TITRES TABLES** est la session. L'avantage, en Client/Serveur, est que plusieurs postes 4D Client peuvent “voir” simultanément votre structure d'une manière différente. Vous pouvez appeler **FIXER TITRES TABLES** autant de fois que vous voulez.

La commande **FIXER TITRES TABLES** est utile dans les cas suivants :

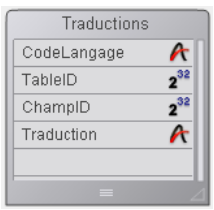
- Traduction dynamique de votre base.
- Affichage des tables dans l'ordre et avec les noms que vous voulez, indépendamment de leur définition.
- Affichage des tables d'une façon qui dépend de l'identité ou des privilèges d'un utilisateur.

Notes :

- **FIXER TITRES TABLES** n'annule pas l'effet de la propriété Invisible d'une table. Si vous avez défini une table en tant qu'invisible au niveau de la structure, elle n'apparaîtra pas en mode Application, même si elle est spécifiée dans **FIXER TITRES TABLES**.
- Les plug-ins accèdent toujours à la structure "virtuelle" telle que définie par cette commande.

Exemple

- Vous développez une application 4D destinée au marché international. Vous avez donc besoin de prendre en compte les nécessités de traduction et de localisation. Pour les éditeurs standard de 4D qui apparaissent en mode Application et vos formulaires utilisant des libellés dynamiques, vous pouvez traiter cette question en utilisant une table *[Traductions]* et quelques méthodes pour créer et utiliser les traductions pour chaque langue que vous voulez.
- Dans votre base, vous créez la table suivante :



- Ensuite, créez la méthode projet *traduire_TABLES_ET_CHAMPS* ci-dessous. Cette méthode analyse la structure de votre base dans la table *[Traductions]* et crée les enregistrements correspondant à la langue passée comme paramètre.

```
` méthode projet traduire_TABLES_ET_CHAMPS
` traduire_TABLES_ET_CHAMPS ( Texte)
` traduire_TABLES_ET_CHAMPS ( CodeLangue )
```

```
C_TEXTE($1) `code de la langue
C_ENTIER LONG($vlTable;$vlChamp)
C_TEXTE($Langue)
$Langue:=$1

Boucle($vlTable;1;Lire numero derniere table) ` Passer sur chaque table
   Si($vlTable#(Table(->[Traductions]))) `Ne pas traduire la table des traductions
 ` Vérifier s'il existe une traduction du nom de la table pour la langue spécifiée
      CHERCHER([Traductions];[Traductions]CodeLangage=$Langue;*)    `langue souhaitée
      CHERCHER([Traductions]; & ;[Traductions]TableID=$vlTable;*)    `numero de table
      CHERCHER([Traductions]; & ;[Traductions]ChampID=0)    `numero de champ = 0 signifie que c'est un nom de table
      Si(Est un numero de table valide($vlTable))    `vérifier que la table existe encore
         Si(Enregistrements trouves([Traductions])=0)
 ` Sinon, créer l'enregistrement
            CREER ENREGISTREMENT([Traductions])
            [Traductions]CodeLangage:=$Langue
            [Traductions]TableID:=$vlTable
            [Traductions]ChampID:=0
 ` Le nom de la table traduit aura besoin d'être saisi
            [Traductions]Traduction:=Nom de la table($vlTable)+" en "+$Langue
            STOCKER ENREGISTREMENT([Traductions])
         Fin de si

         Boucle($vlChamp;1;Lire numero dernier champ($vlTable))
 ` Vérifier s'il existe une traduction pour le nom du champ dans la langue spécifiée
            CHERCHER([Traductions];[Traductions]CodeLangage=$Langue;*)    `langue souhaitée
            CHERCHER([Traductions]; & ;[Traductions]TableID=$vlTable;*)    `numéro de table
            CHERCHER([Traductions]; & ;[Traductions]ChampID=$vlChamp)    `numéro de champ
            Si(Est un numero de champ valide($vlTable;$vlChamp))
               Si(Enregistrements trouves([Traductions])=0)
 ` Sinon, créer l'enregistrement
                  CREER ENREGISTREMENT([Traductions])
                  [Traductions]CodeLangage:=$Langue
                  [Traductions]TableID:=$vlTable
                  [Traductions]ChampID:=$vlChamp
 ` Le nom du champ traduit aura besoin d'être saisi
                  [Traductions]Traduction:=Nom du champ($vlTable;$vlChamp)+" en "+$Langue
                  STOCKER ENREGISTREMENT([Traductions])
               Fin de si
            Sinon
               Si(Enregistrements trouves([Traductions])#0)
 ` si le champ n'existe plus, on supprime la traduction
                  SUPPRIMER ENREGISTREMENT([Traductions])
               Fin de si
            Fin de si
         Fin de boucle
      Sinon
         Si(Enregistrements trouves([Traductions])#0)
 ` si la table n'existe plus, on supprime la traduction
            SUPPRIMER ENREGISTREMENT([Traductions])
         Fin de si
      Fin de si
   Fin de si
Fin de boucle
```

- Si maintenant vous exécutez la ligne suivante, vous pouvez créer autant d'enregistrements qu'il vous faut pour la traduction espagnole de vos tables et champs :

```
traduire_TABLES_ET_CHAMPS("Espagnol")
```

- Une fois que cette ligne de code est appelée, vous pouvez saisir une traduction dans le champ *[Traductions]NomTraduit* pour chacun des nouveaux enregistrements.
- Enfin, chaque fois que vous voulez afficher en espagnol les éditeurs standard de 4D ou les formulaires avec libellés dynamiques, vous exécutez la ligne suivante :

```
TABLES_ET_CHAMPS_LOCALISES("Espagnol")
```

La méthode projet ***TABLES_ET_CHAMPS_LOCALISES*** est la suivante :

```
   ` Méthode objet TABLES_ET_CHAMPS_LOCALISES
   ` TABLES_ET_CHAMPS_LOCALISES ( Texte)
   ` TABLES_ET_CHAMPS_LOCALISES ( CodeLangue )

C_TEXTE($1) `Code de la langue
C_ENTIER LONG($vlTable;$vlChamp)
C_TEXTE($Langue)
C_ENTIER LONG($vlNumTable;$vlNumChamp)
```

```
$Langue:=$1

 `Mise à jour des noms de table
TABLEAU TEXTE($asNoms;0)   ` Initialiser les tableaux pour FIXER TITRES TABLES et FIXER TITRES CHAMPS
TABLEAU ENTIER($aiNuméros;0)
CHERCHER([Traductions];[Traductions]CodeLangage=$Langue;*)
CHERCHER([Traductions]; & ;[Traductions]ChampID=0)    `noms de table donc
SELECTION VERS TABLEAU([Traductions]Traduction;$asNoms;[Traductions]TableID;$aiNuméros)
FIXER TITRES TABLES($asNoms;$aiNuméros)

 `Mise à jour des noms de champs
$vlNumTable:=Lire numero derniere table ` Obtenir le nombre de tables dans la base
Boucle($vlTable;1;$vlNumTable)    ` Passer sur chaque table
   Si(Est un numero de table valide($vlTable))
      CHERCHER([Traductions];[Traductions]CodeLangage=$Langue;*)
      CHERCHER([Traductions]; & ;[Traductions]TableID=$vlTable;*)
      CHERCHER([Traductions]; & ;[Traductions]ChampID#0)    `évite le zero qui sert au nom de la table
      SELECTION VERS TABLEAU([Traductions]Traduction;$asNoms;[Traductions]ChampID;$aiNuméros)
      FIXER TITRES CHAMPS(Table($vlTable)->;$asNoms;$aiNuméros)
   Fin de si
Fin de boucle
```

- Notez que de nouvelles traductions peuvent être effectuées dans la base sans modification de code ni recompilation.

GENERER CLIC

GENERER CLIC (sourisX ; sourisY {; process} {; *})

Paramètre	Type		Description
sourisX	Entier long	⇒	Coordonnée horizontale
sourisY	Entier long	⇒	Coordonnée verticale
process	Entier long	⇒	Numéro de référence du process de destination ou File d'attente des événements de l'application si ce paramètre est omis ou si vous passez 0
*		⇒	Si spécifié, utiliser le système de coordonnées globales Si omis, utiliser le système de coordonnées locales

Description

La commande **GENERER CLIC** simule un clic souris. Elle produit les mêmes effets que lorsque l'utilisateur clique réellement avec le bouton de la souris.

Vous passez les coordonnées horizontale et verticale du clic dans *sourisX* et *sourisY*. Si vous passez le paramètre *, vous exprimez ces coordonnées par rapport à l'écran. Si vous omettez le paramètre *, vous exprimez ces coordonnées par rapport à la fenêtre de premier plan du process dont le numéro est passé dans *process*.

Si vous passez le paramètre *process*, le clic est envoyé au process dont vous avez passé le numéro. Si vous passez *0* (zéro) ou si vous omettez ce paramètre, le clic est envoyé au niveau de l'application et le gestionnaire de 4D l'affectera au process approprié.

GENERER EVENEMENT

GENERER EVENEMENT (quoi ; message ; quand ; sourisX ; sourisY ; modifiers {; process})

Paramètre	Type		Description
quoi	Entier long	⇒	Type d'événement
message	Entier long	⇒	Message de l'événement
quand	Entier long	⇒	Moment de l'événement exprimé en ticks
sourisX	Entier long	⇒	Coordonnée horizontale de la souris
sourisY	Entier long	⇒	Coordonnée verticale de la souris
modifiers	Entier long	⇒	Etat des touches Modifier
process	Entier long	⇒	Numéro de référence du process de destination ou File d'attente des événements de l'application si ce paramètre est omis ou si vous passez 0

Description

La commande **GENERER EVENEMENT** simule un événement (clavier ou souris). Elle produit les mêmes effets que lorsque l'utilisateur agit réellement par l'intermédiaire du clavier ou de la souris.

Vous devez passer une des constantes prédéfinies suivantes dans le paramètre *quoi* :

Constante	Type	Valeur
Bouton souris enfoncé	Entier long	1
Bouton souris relâché	Entier long	2
Répétition touche	Entier long	5
Touche enfoncée	Entier long	3
Touche relâchée	Entier long	4

Si l'événement est lié à la souris, passez *0* (zéro) dans le paramètre *message*. Si l'événement est lié au clavier, passez dans *message* le code du caractère simulé.

Généralement, vous passez la valeur retournée par la fonction **Nombre de ticks** dans *quand*.

Si l'événement est lié à la souris, passez les coordonnées horizontale et verticale du clic dans *sourisX* et *sourisY*.

Dans le paramètre *modifiers*, vous devez passer une constante ou une combinaison de constantes du thème **Evénements (Modifiers)** :

Constante	Type	Valeur	Comment
Bit activation fenêtre	Entier long	0	
Bit bouton souris	Entier long	7	
Bit touche commande	Entier long	8	Touche Ctrl sous Windows, touche Commande sous OS X
Bit touche contrôle	Entier long	12	Touche Ctrl sous OS X, ou clic droit sous Windows et OS X
Bit touche contrôle droite	Entier long	15	
Bit touche majuscule	Entier long	9	Windows et OS X
Bit touche majuscule droite	Entier long	13	
Bit touche option	Entier long	11	Touche Alt (aussi appelée Option sous OS X)
Bit touche option droite	Entier long	14	
Bit touche verrouillage maj	Entier long	10	Windows et OS X
Masque activation fenêtre	Entier long	1	
Masque bouton souris	Entier long	128	
Masque touche commande	Entier long	256	Touche Ctrl sous Windows, touche Commande sous OS X
Masque touche contrôle	Entier long	4096	Touche Ctrl sous OS X, ou clic droit sous Windows et OS X
Masque touche contrôle droite	Entier long	32768	
Masque touche majuscule	Entier long	512	Windows et OS X
Masque touche majuscule droite	Entier long	8192	
Masque touche option	Entier long	2048	Touche Alt (aussi appelée Option sous OS X)
Masque touche option droite	Entier long	16384	
Masque touche verrouillage maj	Entier long	1024	Windows et OS X

Par exemple, pour simuler la touche Majuscule, passez la valeur Bit touche majuscule.

Si vous passez le paramètre *process*, l'événement est envoyé au process dont vous avez passé le numéro. Si vous passez *0* (zéro) ou si vous omettez ce paramètre, l'événement est envoyé au niveau de l'application et le gestionnaire de 4D l'affectera au process approprié.

GENERER FRAPPE CLAVIER

GENERER FRAPPE CLAVIER (code {; modifiers {; process}})

Paramètre	Type		Description
code	Entier long	⇒	Code d'un caractère ou code de touche de fonction
modifiers	Entier long	⇒	Etat des touches Modifier
process	Entier long	⇒	Numéro de référence du process de destination ou File d'attente des événements de l'application si paramètre omis ou égal à 0

Description

La commande **GENERER FRAPPE CLAVIER** simule la frappe d'une touche sur le clavier. Elle produit les mêmes effets que lorsque l'utilisateur tape réellement un caractère au clavier.

Vous passez le code du caractère dans le paramètre *code*.

Si vous n'utilisez pas le paramètre *modifiers*, aucun "modifier" (Majuscule, Option, etc...) n'est simulé. Si vous utilisez le paramètre *modifiers*, vous devez passer une constante ou une combinaison de constantes du thème **Evénements (Modifiers)** :

Constante	Type	Valeur	Comment
Bit activation fenêtre	Entier long	0	
Bit bouton souris	Entier long	7	
Bit touche commande	Entier long	8	Touche Ctrl sous Windows, touche Commande sous OS X
Bit touche contrôle	Entier long	12	Touche Ctrl sous OS X, ou clic droit sous Windows et OS X
Bit touche contrôle droite	Entier long	15	
Bit touche majuscule	Entier long	9	Windows et OS X
Bit touche majuscule droite	Entier long	13	
Bit touche option	Entier long	11	Touche Alt (aussi appelée Option sous OS X)
Bit touche option droite	Entier long	14	
Bit touche verrouillage maj	Entier long	10	Windows et OS X
Masque activation fenêtre	Entier long	1	
Masque bouton souris	Entier long	128	
Masque touche commande	Entier long	256	Touche Ctrl sous Windows, touche Commande sous OS X
Masque touche contrôle	Entier long	4096	Touche Ctrl sous OS X, ou clic droit sous Windows et OS X
Masque touche contrôle droite	Entier long	32768	
Masque touche majuscule	Entier long	512	Windows et OS X
Masque touche majuscule droite	Entier long	8192	
Masque touche option	Entier long	2048	Touche Alt (aussi appelée Option sous OS X)
Masque touche option droite	Entier long	16384	
Masque touche verrouillage maj	Entier long	1024	Windows et OS X

Par exemple, pour simuler la touche Majuscule, passez la valeur Masque touche majuscule.

Si vous passez le paramètre *process*, la frappe clavier est envoyée au process dont le numéro de référence est spécifié. Si vous passez *0* (zéro) dans ce paramètre ou si vous l'omettez, la frappe clavier est envoyée au niveau de l'application et le gestionnaire de 4D l'affectera au process approprié.

Exemple

Reportez-vous à l'exemple de la fonction **Chercher process**.

JOUER SON

JOUER SON (nomObjet {; asynchrone})

Paramètre	Type		Description
nomObjet	Chaîne	⇒	Nom ou chemin de fichier son ou son système Chaîne vide pour stopper un son asynchrone
asynchrone	Entier long	⇒	(Windows) Si passé : exécution asynchrone, si omis : exécution synchrone

Description

La commande **JOUER SON** vous permet de jouer des fichiers de son ou multimedia. Sous OS X, la commande permet également de jouer un son système.

- Pour jouer un fichier, passez son nom et son chemin d'accès dans *nomObjet*. Vous pouvez passer un chemin d'accès complet ou relatif au fichier de structure de la base.
  Les principaux formats de fichiers son et multimedia sont pris en charge : .WAV, .MP3, .AIFF (OS X)... Sous OS X, la commande prend notamment en charge les formats Core Audio.
- (OS X uniquement) Pour jouer un son système, passez directement son nom dans le paramètre *nomObjet*.

Note : Les ressources 'snd', utilisées dans Mac OS 9 et les systèmes plus anciens, ne sont plus prises en charge.

Le paramètre *asynchrone* permet de jouer le son en synchrone ou en asynchrone sous Windows. Synchrone signifie que tous les traitements s'arrêtent jusqu'à ce que le son soit entièrement joué, asynchrone signifie que le traitement ne s'arrête pas et que le son est joué en tâche de fond. Si *asynchrone* est passé et vaut 0 (ou toute valeur numérique), le son est joué en asynchrone. S'il est omis, le son est joué en synchrone.
Note : Sous OS X, le son est toujours asynchrone, que *asynchrone* soit passé ou non.

Pour stopper un son asynchrone, il faut exécuter l'instruction suivante :

```
JOUER SON("";0)
```

Exemple 1

L'exemple suivant montre comment jouer un fichier WAV de votre choix sous Windows :

```
$DocRéf :=Ouvrir document("";"WAV";Mode lecture) //ou MP3...
Si(OK=1)
   FERMER DOCUMENT($DocRéf)
   JOUER SON(Document;0) //exécution asynchrone
Fin de si
```

Exemple 2

Exemple de son système sous OS X :

```
JOUER SON("Submarine.aiff") //Jouer le son système
```

LIRE TITRES CHAMPS

LIRE TITRES CHAMPS (laTable ; titresChamps ; numChamps)

Paramètre	Type		Description
laTable	Table	⇨	Table dont vous souhaitez connaître les noms des champs
titresChamps	Tableau texte	⇦	Noms courants des champs
numChamps	Tableau entier long	⇦	Numéros des champs

Description

La commande **LIRE TITRES CHAMPS** remplit les tableaux *titresChamps* et *numChamps* avec les noms et les numéros des champs de *laTable* désignée. Le contenu des deux tableaux est synchronisé.

Si la commande **FIXER TITRES CHAMPS** a été appelée au cours de la session, **LIRE TITRES CHAMPS** retourne uniquement les noms “modifiés” et les numéros des champs ayant été définis via cette commande.
Sinon, **LIRE TITRES CHAMPS** retourne le nom défini dans la fenêtre de Structure de tous les champs de la base.
Dans les deux cas, la commande ne retourne pas les champs déclarés invisibles.

LIRE TITRES TABLES

LIRE TITRES TABLES (titresTables ; numTables)

Paramètre	Type		Description
titresTables	Tableau texte	⇐	Noms courants des tables
numTables	Tableau entier long	⇐	Numéros des tables

Description

La commande **LIRE TITRES TABLES** remplit les tableaux *titresTables* et *numTables* avec les noms et les numéros des tables de la base définis dans la fenêtre de Structure ou via la commande **FIXER TITRES TABLES**. Le contenu des deux tableaux est synchronisé.

Si la commande **FIXER TITRES TABLES** a été appelée lors de la session, **LIRE TITRES TABLES** retourne uniquement les noms “modifiés” et les numéros des tables ayant été définies via cette commande.
Sinon, **LIRE TITRES TABLES** retourne le nom défini dans la fenêtre de Structure de toutes les tables de la base.
Dans les deux cas, la commande ne retourne pas les tables déclarées invisibles.

Macintosh commande enfoncee

Macintosh commande enfoncee -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Etat de la touche Commande Macintosh ou Etat de la touche Ctrl Windows

Description

Macintosh commande enfoncee retourne Vrai si la touche **Commande** Macintosh est enfoncée.
Note : Lorsqu'elle est appelée sous Windows, la fonction **Macintosh commande enfoncee** retourne Vrai si la touche **Ctrl** Windows est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande **Majuscule enfoncee**.

Macintosh control enfoncee

Macintosh control enfoncee -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Etat de la touche Control du Macintosh

Description

Macintosh control enfoncee retourne **Vrai** si la touche **Control** du Macintosh est enfoncée.

Note : Lorsqu'elle est appelée sous Windows, la fonction **Macintosh control enfoncee** retourne toujours Faux. Cette touche Macintosh n'a pas d'équivalent sous Windows.

Exemple

Reportez-vous à l'exemple de la commande **Majuscule enfoncee**.

Macintosh option enfoncee

Macintosh option enfoncee -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Etat de la touche Option Macintosh ou Etat de la touche Alt Windows

Description

Macintosh option enfoncee retourne Vrai si la touche **Option** Macintosh est enfoncée.
Note : Lorsqu'elle est appelée sous Windows, la fonction **Macintosh option enfoncee** retourne Vrai si la touche **Alt** Windows est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande **Majuscule enfoncee**.

Majuscule enfoncee

Majuscule enfoncee -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Etat de la touche Majuscule

Description

Majuscule enfoncee retourne Vrai si la touche **Majuscule** est enfoncée.

Exemple

La méthode objet du bouton *bUnBouton* effectue des actions différentes en fonction de la ou des touche(s) de modification enfoncée(s) au moment du clic :

```
  ` Méthode objet bUnBouton
Au cas ou
  ` Diverses autres combinaisons de touches peuvent être testées ici
  ` ...
    :(Majuscule enfoncee&Windows Ctrl enfoncee)
  ` Les touches Majuscule et Ctrl Windows (ou Commande Mac OS) sont enfoncées
      FAIRE ACTION1
  ` ...
    :(Majuscule enfoncee)
  ` Seule Majuscule est enfoncée
      FAIRE ACTION2
  ` ...
    :(Windows Ctrl enfoncee)
  ` Seule Ctrl Windows (ou Commande Mac OS) est enfoncée
      FAIRE ACTION3
  ` ...
  ` D'autres touches peuvent être testées individuellement ici
  ` ...
Fin de cas
```

Objet focus

Objet focus -> Résultat

Paramètre	Type	Description
Résultat	Pointeur	Pointeur vers l'objet ayant le focus

Note de compatibilité

Cette commande est conservée pour des raisons de compatibilité uniquement. A compter de la version 12 de 4D, il est conseillé d'utiliser la commande **OBJET Lire pointeur**.

Description

Objet focus retourne un pointeur vers l'objet ayant le focus dans le formulaire courant. Si aucun objet n'a le focus, la commande retourne **Nil**. Vous pouvez utiliser **Objet focus** pour effectuer une action dans un formulaire sans savoir quel objet est actuellement sélectionné. N'oubliez pas auparavant de tester si l'objet est du type voulu, à l'aide de la fonction **Type**.

Note : Lorsqu'elle est utilisée avec une list box, la fonction **Objet focus** retourne un pointeur vers la list box ou la colonne de la list box en fonction du contexte. Pour plus d'informations, reportez-vous à la section **Gestion programmée des objets de type List box**.

Cette commande ne peut pas être utilisée sur les champs dans les sous-formulaires.

Note : Cette commande n'a de sens qu'en cours de saisie. Son utilisation hors de ce contexte génère des messages d'erreur.

Exemple

L'exemple suivant est une méthode objet pour un bouton. Cette méthode passe les données de l'objet courant en majuscules. L'objet doit être de type Texte ou Alpha (type 0 ou 24) :

```
$pointeur :=Objet focus ` Obtenir le pointeur vers le dernier objet
Au cas ou
   :(Nil($pointeur)) ` Aucun objet n'a le focus
      ...
   :((Type($pointeur->)=Est un champ alpha)|(Type($pointeur->)=Est une variable chaîne))
  ` S'il s'agit d'un objet de type Texte ou Alpha
      $pointeur->:=Majusc($pointeur->) ` Mettre les données en majuscules
Fin de cas
```

Pop up menu

Pop up menu (contenu {; parDéfaut {; coordX ; coordY}}) -> Résultat

Paramètre	Type		Description
contenu	Texte	⇒	Définition du texte du menu
parDéfaut	Entier long	⇒	Numéro de l'élément sélectionné par défaut
coordX	Entier long	⇒	Coordonnée X du coin supérieur gauche
coordY	Entier long	⇒	Coordonnée Y du coin supérieur gauche
Résultat	Entier long	⇐	Numéro de l'élément de menu sélectionné

Description

La commande **Pop up menu** fait apparaître un pop up à l'emplacement courant du curseur de la souris ou à l'emplacement défini par les paramètres facultatifs *coordX* et *coordY*.

Selon les règles standard d'interface utilisateur, cette commande doit généralement être appelée en réponse à un clic souris, et lorsque le bouton reste enfoncé un certain laps de temps.

Vous définissez les éléments du pop up menu à l'aide du paramètre *contenu*, de la manière suivante :

- Chaque élément est séparé des autres par un point-virgule (;), *"Elément1;Elément2;Elément3"*.
- Pour inactiver un élément, placez une parenthèse ouvrante "(" dans son libellé.
- Pour définir une ligne de séparation, passez la valeur "-" ou "(-" en tant que libellé.
- Pour définir le style de caractères d'un élément, placez dans son libellé le symbole inférieur à "<" suivi d'une lettre. Voici les différents codes :

<B	Gras
<I	Italique
<U	Souligné
<O	Contours (Mac OS seulement)
<S	Relief (Mac OS seulement)

- Pour associer une coche à un élément, insérez dans son libellé un point d'exclamation "!" suivi du caractère que vous voulez utiliser comme coche.
  - Sous Mac OS, le caractère passé est directement affiché. Pour afficher la coche standard quelle que soit la version ou la langue du système, utilisez l'instruction **Caractere(18)**.
  - Sous Windows, une coche standard est affichée (quel que soit le caractère passé).
- Pour associer une icône à un élément, insérez dans son libellé un accent circonflexe "^" suivi d'un caractère dont le code moins 48 plus 256 (ou plus 208) représente un numéro de ressource d'icône Mac OS.
- Pour ajouter un raccourci clavier à un élément, insérez dans son libellé une barre oblique "/" suivie du caractère de raccourci. Notez que cette dernière option est uniquement informative (aucun raccourci clavier n'active le pop up menu), cependant vous pouvez indiquer un raccourci clavier si l'élément du pop up menu dispose d'une commande équivalente dans la barre de menus principale de votre application.

Astuce : Il est possible de désactiver le mécanisme d'interprétation des caractères spéciaux (!, /, etc.) dans le pop up menu afin, par exemple, de faire figurer ces caractères dans les libellés. Pour cela, il suffit de faire débuter le paramètre *contenu* par l'instruction **Caractere(1)** puis d'utiliser cette instruction comme séparateur :

```
contenu:=Caractere(1)+"1/4"+Caractere(1)+"1/2"+Caractere(1)+"3/4")
```

A noter qu'une fois cette instruction exécutée, il n'est plus possible d'affecter de style ou de raccourcis au pop up menu.

Le paramètre optionnel *parDéfaut* vous permet de définir l'élément du pop up menu sélectionné par défaut lorsque celui-ci apparaît. Passez une valeur située entre 1 et le nombre d'éléments du menu. Si vous ne passez pas ce paramètre, le premier élément du menu sera sélectionné par défaut. Si vous passez également les paramètres *coordX* et *coordY* (cf. ci-dessous), ce paramètre est ignoré.

Les paramètres facultatifs *coordX* et *coordY* permettent de désigner l'emplacement du pop up menu à afficher. Passez respectivement dans *coordX* et *coordY* les coordonnées horizontale et verticale du coin supérieur gauche du menu. Ces coordonnées doivent être exprimées en pixels dans le système de coordonnées local au formulaire courant. Ces deux paramètres doivent être passés ensemble ; si un seul est passé, il est ignoré.
Si vous utilisez les paramètres *coordX* et *coordY*, le paramètre *parDéfaut* est ignoré. Dans ce cas en effet, la souris ne se trouve pas nécessairement au niveau du pop up menu.
Ces paramètres sont utiles notamment pour la gestion des boutons 3D avec pop up menu associé.

Lorsqu'un élément du pop up menu est sélectionné, la commande retourne son numéro, autrement elle retourne zéro.

Note : Utilisez les pop up menus avec un nombre "raisonnable" d'éléments. Si, par exemple, vous voulez afficher plus de 50 éléments, envisagez plutôt d'employer une zone de défilement dans un formulaire.

Exemple

La méthode projet ***MON RACCOURCI*** fait apparaître un pop up menu de navigation :

```
  ` Méthode projet MON RACCOURCI
POSITION SOURIS($vlMouseX;$vlMouseY;$vlBouton)
Si(Macintosh control enfoncee | ($vlBouton=2))
   $vtItems:="A propos de cette base...<I;(-;!-Autres options;(-"
   Boucle($vlTable;1;Lire numero derniere table)
      Si(Est un numero de table valide($vlTable))
         $vtItems:=$vtItems+";"+Nom de la table($vlTable)
      Fin de si
   Fin de boucle
   $vlChoixUtilisateur:=Pop up menu($vtItems)
   Au cas ou
      :($vlChoixUtilisateur=1)
```

```
    ` Afficher les informations
        :($vlChoixUtilisateur=2)
    ` Afficher les options
        Sinon
            Si($vlChoixUtilisateur>0)
    ` Aller à la table dont le numéro est $vlChoixUtilisateur-4
            Fin de si
      Fin de cas
  Fin de si
```

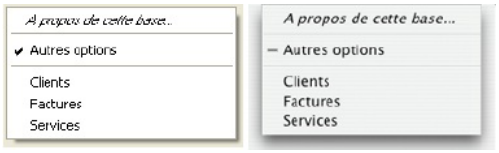
Cette méthode projet peut être appelée d'une des manières suivantes :

- depuis la méthode d'un objet réagissant à un clic souris, et n'attendant pas que le bouton soit relâché (par exemple un bouton invisible),
- depuis un process qui “épie” les événements et communique avec les autres process,
- depuis une méthode de gestion d'événements installée par la commande **APPELER SUR ERREUR**.

Dans les deux derniers cas, il n'est pas nécessaire que le clic se produise dans un objet de formulaire. C'est l'un des avantages de la commande **Pop up menu**. Généralement, les pop up menus sont affichés par l'intermédiaire d'objets de formulaire. Avec **Pop up menu**, vous pouvez faire apparaître un pop up menu n'importe où.

Le pop up menu s'affiche sous Windows lorsque l'utilisateur appuie sur le **bouton droit** de la souris, et sous Mac OS lorsqu'il utilise la combinaison **Control+clic**. Notez cependant que la méthode ci-dessus ne teste pas le clic souris, c'est la méthode appelante qui en est chargée.

Voici le pop up menu tel qu'il s'affiche sous Windows (à gauche) et sous Mac OS (à droite). Notez la coche standard de la version Windows :



POSITION SOURIS

POSITION SOURIS (sourisX ; sourisY ; boutonSouris {; *})

Paramètre	Type		Description
sourisX	Réel	⇐	Coordonnée horizontale de la souris
sourisY	Réel	⇐	Coordonnée verticale de la souris
boutonSouris	Entier long	⇐	Etat du bouton de la souris : 0 = Bouton relâché 1 = Bouton enfoncé 2 = Bouton droit enfoncé 3 = Les deux boutons enfoncés
*	Opérateur	⇒	Si spécifié, utiliser le système de coordonnées globales Si omis, utiliser le système de coordonnées locales

Description

La commande **POSITION SOURIS** retourne l'état courant de la souris.

Les coordonnées horizontale et verticale sont retournées dans les paramètres *sourisX* et *sourisY*. Si vous passez le paramètre *, ces coordonnées sont exprimées par rapport à l'écran. Si vous ne passez pas le paramètre *, vous exprimez ces coordonnées par rapport à la fenêtre du formulaire courant (s'il y en a un) du process courant.

Le paramètre *boutonSouris* retourne l'état du ou des bouton(s) de la souris, comme décrit ci-dessus dans le tableau des paramètres.

Note : Les valeurs 2 et 3 peuvent être retournées sous Mac OS X à compter de la version 10.2.5 uniquement.

Exemple

Reportez-vous à l'exemple de la commande **Pop up menu**.

REDESSINER

REDESSINER (objet)

Paramètre	Type		Description
objet	Objet de formulaire	⇨	Table de laquelle redessiner le sous-formulaire ou Champ duquel redessiner la zone ou Variable de laquelle redessiner la zone ou List box à mettre à jour

Description

Lorsque vous modifiez par programmation le contenu d'un champ affiché dans un sous-formulaire, vous devez exécuter la commande **REDESSINER** pour vous assurer que le formulaire est correctement mis à jour.

Dans le contexte des list box en mode sélection, l'instruction **REDESSINER** appliquée à un objet de type list box provoque la mise à jour des données affichées dans l'objet. Cette instruction doit être appelée typiquement après une modification des données dans les enregistrements de la sélection.

Verrouillage majuscule enfoncee

Verrouillage majuscule enfoncee -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Etat de la touche Verrouillage Majuscule

Description

Verrouillage majuscule enfoncee retourne Vrai si la touche **Verrouillage Majuscule** est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande **Majuscule enfoncee**.

Windows Alt enfoncee

Windows Alt enfoncee -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Etat de la touche Windows Alt ou Etat de la touche Macintosh Option

Description

Windows Alt enfoncee retourne Vrai si la touche **Alt** Windows est enfoncée.
Note : Lorsqu'elle est appelée sous Mac OS, **Windows Alt enfoncee** retourne Vrai si la touche Macintosh **Option** est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande **Majuscule enfoncee**.

Windows Ctrl enfoncee

Windows Ctrl enfoncee -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Etat de la touche Ctrl Windows ou Etat de la touche Commande Macintosh

Description

Windows Ctrl enfoncee retourne Vrai si la touche **Ctrl** Windows est enfoncée.
Note : Lorsqu'elle est appelée sous Mac OS, **Windows Ctrl enfoncee** retourne Vrai si la touche Macintosh **Commande** est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande **Majuscule enfoncee**.

_o_FIXER INTERFACE

_o_FIXER INTERFACE (interface)

Paramètre	Type		Description
interface	Entier long	⇒	Nouvelle interface de plate-forme : -1 Plate-forme automatique 0 Mac OS 7 1 Windows 3.11, NT 3.51 2 Windows 9x 3 Mac OS 9 4 Thème Mac

Note de compatibilité

L'interface de plate-forme est gérée automatiquement par 4D. Cette commande est désormais ignorée et ne doit plus être utilisée.

_o_INVERSER FOND

_o_INVERSER FOND ({* ;} objetTexte)

Paramètre	Type		Description
*	Opérateur	⇨	Si spécifié, on passe le nom de l'objet (chaîne) Si omis, on passe un champ ou une variable
objetTexte	Variable, Champ	⇨	Variable ou champ de type texte dont le fond doit être inversé

Commande obsolète

Cette commande n'est plus prise en charge dans 4D.

_o_Lire interface

_o_Lire interface -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Interface de plate-forme en cours d'utilisation

Note de compatibilité

L'interface de plate-forme est désormais gérée automatiquement par 4D. Cette commande ne retourne plus de valeur significative ne doit plus être utilisée.

Interruptions

- APPELER SUR ERREUR
- APPELER SUR EVENEMENT
- ASSERT
- Asserted
- FILTRER EVENEMENT
- FIXER ACTIVATION ASSERTIONS
- Lire activation assertions
- LIRE PILE DERNIERE ERREUR
- Methode appelee sur erreur
- Methode appelee sur evenement
- STOP

APPELER SUR ERREUR

APPELER SUR ERREUR (méthodErreur)

Paramètre	Type		Description
méthodErreur	Chaîne	➡	Méthode de gestion d'erreur à appeler ou Chaîne vide pour désinstaller la méthode

Description

APPELER SUR ERREUR installe la méthode projet dont le nom est passé dans *méthodErreur* comme méthode d'interception des erreurs — aussi appelée méthode de gestion des erreurs.

Après l'installation, 4D appelle cette méthode lorsqu'une erreur se produit lors de l'exécution d'une commande du langage 4D.

La portée de cette commande est le process courant. Il ne peut y avoir qu'une seule méthode de gestion des erreurs par process, mais il peut exister différentes méthodes de gestions d'erreurs pour plusieurs process.

Notes :

- Dans le contexte de l'utilisation de composants, cette commande s'applique à la base courante uniquement. Si une erreur est générée depuis un composant, *méthodErreur* n'est pas appelée dans la base hôte.
- Si **APPELER SUR ERREUR** est appelée depuis un process dont vous avez demandé l'exécution en préemptif (en mode compilé 64 bits), le compilateur vérifiera le caractère thread-safe de *méthodErreur* et retournera des erreurs si elle n'est pas compatible avec le mode préemptif. Pour plus d'informations, reportez-vous à la section **Process 4D préemptifs**.

Pour désinstaller une méthode de gestion des erreurs, appelez de nouveau **APPELER SUR ERREUR** et passez une chaîne vide dans *méthodErreur*.

Vous pouvez identifier les erreurs en lisant la variable système *Error*, qui contient le code de l'erreur. Les codes d'erreurs retournés par 4D sont traités dans les sections **Codes d'erreurs**. Reportez-vous par exemple à la section **Erreurs de syntaxe (1 -> 81)**. La variable *Error* n'est définie qu'à l'intérieur de la méthode de gestion des erreurs ; si vous souhaitez que le code soit accessible dans la méthode ayant provoqué l'erreur, copiez la variable *Error* dans votre propre variable process. Vous pouvez également accéder aux variables système *Error method*, *Error line* et *Error formula* contenant respectivement le nom de la méthode, le numéro de ligne et le texte de la formule à l'origine de l'erreur (cf. paragraphe **Error, Error method, Error line**).

Vous pouvez obtenir la séquence d'erreurs à l'origine de l'interruption (la "pile" d'erreurs) à l'aide de la commande **LIRE PILE DERNIERE ERREUR**.

La méthode de gestion des erreurs doit généralement traiter les erreurs de manière appropriée ou afficher un message d'erreur à l'utilisateur. Les erreurs peuvent être générées lors de traitements effectués sur :

- Le moteur de base de données de 4D ; par exemple, lorsque la sauvegarde d'un enregistrement provoquerait la violation d'une règle de trigger.
- L'environnement de 4D ; par exemple, lorsque vous n'avez pas assez de mémoire pour remplir un tableau.
- Le système d'exploitation sur lequel la base est lancée ; par exemple, disque plein ou erreurs d'entrée/sortie.

La commande **STOP** peut être utilisée pour stopper le traitement. Si vous n'appelez pas **STOP** dans la méthode installée, 4D retourne à la méthode interrompue et reprend son exécution. Utilisez la commande **STOP** lorsque l'exécution ne peut se poursuivre.

Si une erreur se produit dans la méthode de gestion d'erreurs elle-même, 4D reprend le contrôle de la gestion des erreurs. En conséquence, assurez-vous que la méthode de gestion des erreurs installée ne puisse pas elle-même générer d'erreur. Aussi, vous ne pouvez pas utiliser la commande **APPELER SUR ERREUR** dans une méthode de gestion des erreurs.

Exemple 1

La méthode projet suivante tente de créer un document dont le nom est reçu en paramètre et retourne 0 (zéro) ou un code d'erreur si le document n'a pas pu être créé :

```
  ` Méthode projet Créer doc
  ` Créer doc ( Chaîne ; Pointeur ) -> Entier long
  ` Créer doc ( NomDoc ; ->DocRef ) -> Code d'erreur résultant

gError:=0
APPELER SUR ERREUR("IO TRAITEMENT ERREURS")
$2->:=Creer document($1)
APPELER SUR ERREUR("")
$0:=gError
```

La méthode projet ***IO TRAITEMENT ERREURS*** est la suivante :

```
  ` Méthode projet IO TRAITEMENT ERREURS
```

```
gError:=Error ` Simple copie du code d'erreur dans la variable process gError
```

Notez l'utilisation de la variable process *gError* pour récupérer le code d'erreur dans la méthode en train de s'exécuter. Une fois que ces méthodes sont présentes dans votre base, vous pouvez écrire par exemple :

```
  ` ...
C_HEURE(vhDocRef)
$vlErrCode:=Créer doc($vsDocumentNom;->vhDocRef)
Si($vlErrCode=0)
  ` ...
   FERMER DOCUMENT($vlErrCode)
Sinon
   ALERTE("Le document n'a pas pu être créé, erreur d'E/S "+Chaine($vlErrCode))
```

```
Fin de si
```

Exemple 2

Reportez-vous à l'exemple de la section **Tableaux et mémoire**.

Exemple 3

Alors que vous implémentez un ensemble complexe d'opérations, vous pouvez terminer avec de multiples sous-routines qui nécessitent différentes méthodes de gestion des erreurs. Comme ne pouvez avoir qu'une seule méthode à la fois de gestion des erreurs par process, vous devez soit repérer la méthode courante à chaque fois que vous appelez **APPELER SUR ERREUR**, soit utiliser une variable tableau process (ici *tabErrorMethod*) pour “empiler” les méthodes de gestion d'erreur ainsi qu'une méthode projet (ici ***APPEL SUR ERR***) pour les installer et les désinstaller. Le tableau doit être initialisé au tout début de l'exécution du process :

```
// N'oubliez pas d'initialiser le tableau au début
// de la méthode de gestion du process
TABLEAU ALPHA(63;tabErrorMethod;0)
```

Voici la méthode personnalisée ***APPEL SUR ERR*** :

```
// Méthode projet APPEL SUR ERR
// APPEL SUR ERR { ( Chaîne ) }
// APPEL SUR ERR { ( Nom de la méthode ) }

C_ALPHA(63;$1;$ErrorMethod)
C_ENTIER LONG($vlElem)

Si(Nombre de paramètres>0)
   $ErrorMethod:=$1
Sinon
   $ErrorMethod:=""
Fin de si

Si($ErrorMethod#"")
   C_ENTIER LONG(gError)
   gError:=0
   $vlElem:=1+Taille tableau(tabErrorMethod)
   INSERER DANS TABLEAU(tabErrorMethod;$vlElem)
   tabErrorMethod{$vlElem}:=$1
   APPELER SUR ERREUR($1)
Sinon
   APPELER SUR ERREUR("")
   $vlElem:=Taille tableau(tabErrorMethod)
   Si($vlElem>0)
      SUPPRIMER DANS TABLEAU(tabErrorMethod;$vlElem)
      Si($vlElem>1)
         APPELER SUR ERREUR(tabErrorMethod{$vlElem-1})
      Fin de si
   Fin de si
Fin de si
```

Vous pouvez alors l'appeler de la manière suivante :

```
gError:=0
APPEL SUR ERR("ERREURS ES") //Installe la méthode de gestion d'erreurs ERREURS ES
 // ...
APPEL SUR ERR("TOUTES ERREURS") //Installe la méthode de gestion d'erreurs TOUTES ERREURS
 // ...
APPEL SUR ERR //Désinstalle la méthode de gestion d'erreurs TOUTES ERREURS et réinstalle ERREURS ES
 // ...
APPEL SUR ERR //Désinstalle la méthode de gestion d'erreurs ERREURS ES
 // ...
```

Exemple 4

La méthode de gestion d'erreurs suivante ignore les interruptions de l'utilisateur et affiche le texte de l'erreur :

```
//Méthode projet Montrer_seulement_erreurs
Si(Error#1006) //ce n'est pas une interruption utilisateur
   ALERTE("L'erreur "+Chaine(Error)+" s'est produite. Le code en cause est : \""+Error formula+"\"")
Fin de si
```

APPELER SUR EVENEMENT

APPELER SUR EVENEMENT (méthodeEvén {; nomProcess})

Paramètre	Type		Description
méthodeEvén	Chaîne	⇒	Méthode d'événement à appeler ou Chaîne vide pour arrêter l'interception des événements
nomProcess	Chaîne	⇒	Nom de process

Description

APPELER SUR EVENEMENT installe la méthode dont le nom est passé dans *méthodeEvén* comme méthode de gestion des événements.

Conseil : Cette commande nécessite un niveau de connaissances avancé en programmation. Généralement, vous n'avez pas besoin d'appeler **APPELER SUR EVENEMENT** pour traiter les événements. Lorsque vous utilisez des formulaires, 4D gère pour vous les événements et les retourne aux formulaires et objets appropriés.

Astuce : Les commandes telles que **POSITION SOURIS**, **Majuscule enfoncee**, etc., permettent de récupérer des informations sur les événements. Ces commandes, dans une certaine mesure, peuvent être appelées depuis les méthodes objet pour traiter les informations dont vous avez besoin. Elles peuvent ainsi vous épargner l'écriture d'un algorithme basé sur une structure du type **APPELER SUR EVENEMENT**.

La portée de cette commande est la session de travail. Par défaut, la méthode est exécutée dans un process local séparé. Vous ne pouvez avoir qu'une méthode de gestion d'événement à la fois. Pour désinstaller une méthode de gestion d'événement, appelez de nouveau **APPELER SUR EVENEMENT** et passez une chaîne vide dans *méthodeEvén*.

Comme la méthode de gestion d'événement tourne dans process séparé, *méthodeEvén* est toujours active, même si aucune méthode 4D n'est en cours d'exécution. Après l'installation, 4D appelle la méthode *méthodeEvén* dès qu'un événement survient. Un événement peut être un clic souris ou la frappe d'une touche.

Le paramètre optionnel *nomProcess* permet de donner un nom au process créé par **APPELER SUR EVENEMENT**. Si *nomProcess* commence par le symbole dollar (*$*), *nomProcess* est un process local, ce dont vous aurez généralement besoin. Si vous ne passez pas le paramètre *nomProcess*, 4D crée par défaut un process local nommé *$Gestionnaire d'événement*.

ATTENTION : Soyez prudent lors de l'écriture d'une méthode de gestion d'événement. N'appelez pas de commande générant un événement, sinon vous risquez de ne plus pouvoir sortir de la méthode. La combinaison de touches **Ctrl+Maj+Retour Arrière** (sous Windows) ou **Commande+Maj+Control+Retour Arrière** (sous Mac) permet de tuer le process d'événement. Cette combinaison vous permet de sortir d'une méthode de gestion d'événement devenue incontrôlable.

Dans la méthode de gestion d'événement, vous pouvez lire les variables système suivantes : **MouseDown**, **KeyCode**, **Modifiers**, **MouseX**, **MouseY** et **MouseProc**. Notez que ces variables sont des variables process. Leur portée est donc le process de gestion d'événements. Copiez-les dans des variables interprocess si vous souhaitez que leurs valeurs soient disponibles dans un autre process.

- La variable système **MouseDown** contient 1 s'il y a eu un clic souris, 0 sinon.
- La variable système **KeyCode** contient le code du caractère tapé au clavier, ou le code d'une touche de fonction. Référez-vous aux sections **Codes Unicode** and **EXPORTER TEXTE** qui listent les codes de caractères utilisés par 4D, ainsi qu'à la section **Codes des touches de fonction**. 4D fournit des constantes prédéfinies pour les principaux codes ASCII et touches de fonctions. Vous pouvez les visualiser à l'aide la fenêtre de l'Explorateur, dans les thèmes correspondants.
- La variable système **Modifiers** permet de savoir si une touche de modification (*modifier*) était enfoncée au moment où l'événement s'est produit. Les touches suivantes peuvent être détectées :

Plate-forme	Modifiers
Windows	Maj, Verrouillage des majuscules, Alt, Ctrl
Macintosh	Maj, Verrouillage des majuscules, Alt (ou Option), Ctrl, Commande

Isolément, les touches *modifiers* ne génèrent pas d'événement. Pour cela, une autre touche ou le bouton de la souris doit également être enfoncé(e). La variable **Modifiers**, de type Entier long, contient un champ de bits. 4D fournit des constantes désignant la position ou le masque des bits pour chaque touche de modification. Lorsque, par exemple, vous voulez détecter si la touche Majuscule était enfoncée pour l'événement, vous pouvez écrire indifféremment :

```
Si(Modifiers?? Bit touche majuscule) //Si la touche Majuscule était enfoncée
```

ou :

```
Si((Modifiers & Masque touche majuscule)#0) //Si la touche Majuscule était enfoncée
```

Les constantes à utiliser en fonction du *modifier* à tester et de la plate-forme sont les suivantes, placées dans le thème **Evénements (Modifiers)** :

Modifier	Constante
Majuscule	Bit touche majuscule / Masque touche majuscule
Verr. majuscule	Bit touche verrouillage maj / Masque touche verrouillage maj
Alt (aussi appelée Option sous OS X)	Bit touche option / Masque touche option
Ctrl sous Windows	Bit touche commande / Masque touche commande
Ctrl sous OS X	Bit touche contrôle / Masque touche contrôle
Commande sous OS X	Bit touche commande / Masque touche commande
Clic droit	Bit touche contrôle / Masque touche contrôle

- Les variables systèmes **MouseX** et **MouseY** contiennent les coordonnées horizontale et verticale du clic souris, exprimées dans le système de coordonnées locales de la fenêtre dans laquelle le clic s'est produit. L'angle supérieur gauche de la fenêtre a les coordonnées 0,0. Ces variables n'ont de signification que lorsqu'un clic souris a eu lieu.
- La variable système **MouseProc** contient le numéro de référence du process dans lequel le clic souris s'est produit.

Note : Les variables système **MouseDown**, **KeyCode**, **Modifiers**, **MouseX**, **MouseY** et **MouseProc** ne contiennent des valeurs significatives que dans

une méthode de gestion d'événement installée par **APPELER SUR EVENEMENT**.

Exemple

L'exemple suivant annule l'impression si l'utilisateur appuie sur les touches **Ctrl+.** (**Commande+.** sous Mac OS). En premier lieu, la méthode de gestion des événements est installée. Ensuite, un message s'affiche, indiquant que l'impression a été annulée. Si la variable interprocess *◊vbOnStoppe* est égale à **Vrai** dans la méthode de gestion d'événement, une boîte de dialogue d'alerte s'affiche pour indiquer à l'utilisateur le nombre d'enregistrements qui viennent de s'imprimer. Enfin, la méthode de gestion d'événement est désinstallée :

```
UTILISER PARAMETRES IMPRESSION
Si(OK=1)
   ◊vbOnStoppe:=Faux
   APPELER SUR EVENEMENT("GESTION EVENEMENTS")
   TOUT SELECTIONNER([Personnes])
   MESSAGE("Pour interrompre l'impression, appuyez sur Ctrl+point.")
   $NbEnregistrements:=Enregistrements trouves([Personnes])
   Boucle($Enrg;1;$NbEnregistrements)
      Si(◊vbOnStoppe)
         ALERTE("L'impression a été annulée à l'enregistrement "+Chaine($Enrg)+" sur "+Chaine($NbEnregistrements))
         $Enrg:=$NbEnregistrements+1
      Sinon
         Imprimer ligne([Personnes];"Etat")
      Fin de si
   Fin de boucle
   SAUT DE PAGE
   APPELER SUR EVENEMENT("") ` Désinstallation de la méthode d'appel sur événement
Fin de si
```

La méthode de gestion d'événement teste si la combinaison de touches **Ctrl+.** (**Commande+.**) a été employée et met la variable interprocess *◊vbOnStoppe* à **Vrai** :

```
  ` Méthode projet GESTION EVENEMENTS
Si((Modifiers?? Bit touche commande) & (KeyCode=Point))
   CONFIRMER("Voulez-vous vraiment annuler l'impression ?")
   Si(OK=1)
      ◊vbOnStoppe:=Vrai
      FILTRER EVENEMENT ` N'oubliez pas cet appel sinon 4D traitera aussi cet événement
   Fin de si
Fin de si
```

Notez que **APPELER SUR EVENEMENT** est utilisé dans cet exemple car un état spécial est imprimé à l'aide des commandes **UTILISER PARAMETRES IMPRESSION**, **Imprimer ligne** et **SAUT DE PAGE** dans une structure de type **Boucle...Fin de boucle**.

Lorsque vous imprimez un état à l'aide la commande **IMPRIMER SELECTION**, vous n'avez pas besoin de gérer les événements permettant à l'utilisateur d'interrompre l'impression, **IMPRIMER SELECTION** le fait pour vous.

ASSERT

ASSERT (expressionBool {; texteMessage})

Paramètre	Type		Description
expressionBool	Booléen	➡	Expression booléenne
texteMessage	Texte	➡	Texte du message d'erreur

Description

La commande **ASSERT** évalue l'assertion *expressionBool* passée en paramètre et, si elle retourne faux, interrompt l'exécution du code en affichant une erreur. La commande fonctionne en mode interprété et en mode compilé.

Si l'expression est vraie, il ne se passe rien. Si l'expression est fausse, la commande déclenche l'erreur -10518 et affiche par défaut le texte de l'assertion précédé du message "Fausse assertion :". Vous pouvez intercepter cette erreur via une méthode installée par la commande **APPELER SUR ERREUR**, afin par exemple d'alimenter un fichier d'historique.

Optionnellement, vous pouvez passer un paramètre *texteMessage* afin d'afficher un message d'erreur personnalisé au lieu du texte de l'assertion.

Une assertion est une instruction insérée dans le code d'une méthode et chargée de détecter des éventuelles anomalies au cours de son exécution. Le principe consiste à vérifier qu'une expression est vraie à un instant donné et, dans le cas contraire, produire une exception. Les assertions sont surtout utilisées pour détecter des cas qui ne devraient jamais arriver en temps normal. Elles servent donc essentiellement à détecter des bogues de programmation. Il est possible d'activer ou de désactiver globalement toutes les assertions d'une application (par exemple en fonction du type de version) via la commande **FIXER ACTIVATION ASSERTIONS**. Pour plus d'informations sur les assertions en programmation, reportez-vous à l'article (en anglais) qui leur est consacré sur Wikipedia : http://en.wikipedia.org/wiki/Assertion_(computing)

Exemple 1

Avant d'effectuer des opérations sur un enregistrement, le développeur souhaite s'assurer qu'il est bien chargé en lecture écriture :

```
LECTURE ECRITURE([Table 1])
CHARGER ENREGISTREMENT([Table 1])
ASSERT(Non(Enregistrement verrouille([Table 1])))  //déclenche l'erreur  -10518  si l'enregistrement est
verrouillé
```

Exemple 2

Une assertion peut permettre de tester les paramètres passés à une méthode projet pour détecter des valeurs aberrantes. Dans cet exemple, un message d'alerte personnalisé est utilisé.

```
 // Méthode qui retourne le numéro d'un client en fonction de son nom passé dans  $1
C_TEXTE($1) // Nom du client
ASSERT($1#"";"Recherche d'un nom de client vide")
    // Un nom vide dans ce cas est une valeur aberrante
    // Si assertion fausse, affichera dans la boîte de dialogue d'erreur :
    // "Fausse assertion : Recherche d'un nom de client vide"
```

Asserted

Asserted (expressionBool {; texteMessage}) -> Résultat

Paramètre	Type		Description
expressionBool	Booléen	➡	Expression booléenne
texteMessage	Texte	➡	Texte du message d'erreur
Résultat	Booléen	⮌	Résultat de l'évaluation d'expressionBool

Description

La commande **Asserted** a un fonctionnement semblable à celui de la commande **ASSERT**, à la différence près qu'elle retourne une valeur issue de l'évaluation du paramètre *expressionBool*. Elle permet donc d'utiliser une assertion lors de l'évaluation d'une condition (cf. exemple). Pour plus d'informations sur le fonctionnement des assertions et sur les paramètres de cette commande, reportez-vous à la description de la commande **ASSERT**.

Asserted accepte une expression booléenne en paramètre et retourne le résultat de l'évaluation de cette expression. Si l'expression est fausse et si les assertions sont activées (cf. commande **FIXER ACTIVATION ASSERTIONS**), l'erreur -10518 est générée, exactement comme pour la commande **ASSERT**. Si les assertions sont inactivées, **Asserted** retourne simplement le résultat de l'expression qui lui est passée sans déclencher d'erreur.

Note : Comme la commande **ASSERT**, **Asserted** fonctionne en mode interprété et en mode compilé.

Exemple

Insertion d'une assertion dans l'évaluation d'une expression :

```
LECTURE ECRITURE([Table 1])
CHARGER ENREGISTREMENT([Table 1])
Si(Asserted(Non(Enregistrement verrouille([Table 1]))))
      // Ce code déclenche l'erreur  -10518  si l'enregistrement est verrouillé
      ...
Fin de si
```

FILTRER EVENEMENT

FILTRER EVENEMENT
Ne requiert pas de paramètre

Description

FILTRER EVENEMENT doit être appelée à l'intérieur d'une méthode de gestion d'événements installée par **APPELER SUR EVENEMENT**.

Lorsqu'une méthode de gestion d'événements appelle la commande **FILTRER EVENEMENT**, l'événement courant n'est pas passé à 4D.

Cette commande vous permet d'effacer l'événement courant (i.e. clic, frappe clavier) de la séquence d'événements, de manière à ce que 4D n'effectue pas de traitement sur l'événement que vous provoquez dans la méthode de gestion d'événements.

ATTENTION : Evitez de créer une méthode de gestion d'événement appelant uniquement **FILTRER EVENEMENT** car TOUS les événements vont être ignorés par 4D. Si vous vous retrouvez dans un tel cas, vous pouvez sortir de la méthode en tapant **Ctrl**+**Maj**+**Retour Arrière** (sous Windows) ou **Commande**+**Option**+**Maj**+**Contrôle**+**Retour Arrière** (sous Mac OS). Dans ce cas, le process de gestion d'événement est converti en process normal n'interceptant plus aucun événement.

Cas particulier : La commande **FILTRER EVENEMENT** peut également être utilisée au sein d'une méthode de formulaire sortie standard, lorsque le formulaire est affiché par l'intermédiaire des commandes **VISUALISER SELECTION** ou **MODIFIER SELECTION**. Dans ce cas précis, la commande **FILTRER EVENEMENT** permet de filtrer les double-clics sur les enregistrements (et ainsi, exécuter d'autres actions que l'ouverture des enregistrements en mode page).
Pour cela, il vous suffit de placer dans la méthode du formulaire sortie les lignes suivantes :

```
Si(Evenement formulaire=Sur double clic souris)
   FILTRER EVENEMENT
   ... `Traiter le double-clic
Fin de si
```

Exemple

Référez-vous à l'exemple d'**APPELER SUR EVENEMENT**.

FIXER ACTIVATION ASSERTIONS

FIXER ACTIVATION ASSERTIONS (asserts {; *})

Paramètre	Type		Description
asserts	Booléen	⇒	Vrai = activer les assertions, Faux = désactiver les assertions
*	Opérateur	⇒	Si omis = la commande s'applique à l'ensemble des process, Si passé = la commande s'applique au process courant uniquement

Description

La commande **FIXER ACTIVATION ASSERTIONS** permet de désactiver ou de réactiver les assertions éventuellement insérées dans le code 4D de l'application. Pour plus d'informations sur les assertions, reportez-vous à la description de la commande **ASSERT**.

Par défaut, les assertions ajoutées dans le programme sont actives, en mode interprété et en mode compilé. Cette commande est utile pour les désactiver car leur évaluation peut parfois être coûteuse en temps d'exécution et vous pouvez aussi souhaiter les masquer pour l'utilisateur final de l'application. Typiquement, la commande **FIXER ACTIVATION ASSERTIONS** pourra être utilisée dans la **Méthode base Sur ouverture** afin d'activer ou non les assertions suivant que l'application est en mode "Test" ou en mode "Production".

Par défaut, la commande **FIXER ACTIVATION ASSERTIONS** agit sur tous les process de l'application déjà créés ou créés par la suite. Pour restreindre l'effet de la commande au process courant uniquement, passez le paramètre *.

A noter que lorsque les assertions sont désactivées, les expressions passées aux commandes **ASSERT** ne sont plus évaluées. Les lignes de code appelant **ASSERT** n'ont alors plus aucun effet sur le fonctionnement de l'application, ni en termes de comportement ni en terme de performances.

Exemple

Désactivation globale des assertions :

```
FIXER ACTIVATION ASSERTIONS(Faux)
ASSERT(MéthodeTest) // MéthodeTest ne sera pas appelée car les asserts sont désactivés
```

Lire activation assertions

Lire activation assertions -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai = les assertions sont activées, Faux = les assertions sont inactivées

Description

La commande **Lire activation assertions** retourne Vrai ou Faux suivant que les assertions sont actives ou non dans le process courant. Pour plus d'informations sur les assertions, reportez-vous à la description de la commande **ASSERT**.

Par défaut, les assertions sont actives mais elles peuvent avoir été désactivées à l'aide de la commande **FIXER ACTIVATION ASSERTIONS**.

LIRE PILE DERNIERE ERREUR

LIRE PILE DERNIERE ERREUR (tabCodes ; tabCompInternes ; tabLibellés)

Paramètre	Type		Description
tabCodes	Tableau entier long	⇦	Tableau de numéros d'erreurs
tabCompInternes	Tableau chaîne	⇦	Tableau de codes de composants internes
tabLibellés	Tableau chaîne	⇦	Tableau de libellés d'erreurs

Description

La commande **LIRE PILE DERNIERE ERREUR** retourne les informations relatives à la “pile” d'erreurs courante de l’application 4D. Lorsqu’une instruction 4D provoque une erreur, la pile d’erreurs courante contient la description de l’erreur ainsi que les éventuelles erreurs générées en cascade. Par exemple l’erreur du type "disque saturé" entraîne une erreur d'écriture dans le fichier puis une erreur dans la commande de sauvegarde d’enregistrements : la pile contient alors trois erreurs. Si la dernière instruction 4D n’a pas généré d’erreur, la pile d’erreurs courante est vide.

Cette commande générique permet de traiter tous les types d’erreurs pouvant se produire dans l'application 4D.

Note : Toutefois, pour obtenir des informations détaillées relatives aux erreurs générées par une source ODBC, il est nécessaire d’utiliser la commande **SQL LIRE DERNIERE ERREUR**.

La commande **LIRE PILE DERNIERE ERREUR** doit être appelée dans le cadre d’une méthode d’appel sur erreur installée par la commande **APPELER SUR ERREUR**.

Les informations sont retournées sous la forme de trois tableaux synchronisés :

- *tabCodes* : ce tableau reçoit la liste des codes d’erreurs générés.
- *tabCompInternes* : ce tableau contient les codes des composants internes associés à chaque erreur.
- *tabLibellés* : ce tableau contient les libellés de chaque erreur.

La liste des codes d’erreurs et de leurs libellés est fournie dans les sections du thème "**Codes d'erreurs**".

Methode appelee sur erreur

Methode appelee sur erreur -> Résultat

Paramètre	Type		Description
Résultat	Chaîne		Nom de la méthode d'appel sur erreur

Description

La commande **Methode appelee sur erreur** retourne le nom de la méthode installée par la commande **APPELER SUR ERREUR** pour le process courant.

Si aucune méthode d'appel sur erreur n'a été installée, une chaîne vide ("") est retournée.

Exemple

Cette commande est particulièrement utile dans le cadre des composants, car elle permet de changer temporairement puis de rétablir les méthodes d'interception d'erreurs :

```
$methCourante:=Methode appelee sur erreur
APPELER SUR ERREUR("NouvelleMéthode")
  ` Si le document ne peut être ouvert, une erreur est générée
$ref:=Ouvrir document("MonDocument")
  ` Réinstallation de la méthode précédente
APPELER SUR ERREUR($methCourante)
```

Methode appelee sur evenement

Methode appelee sur evenement -> Résultat

Paramètre	Type		Description
Résultat	Chaîne		Nom de la méthode d'appel sur evenement

Description

La commande **Methode appelee sur evenement** retourne le nom de la méthode installée par la commande **APPELER SUR EVENEMENT**.
Si aucune méthode d'appel sur événement n'a été installée, une chaîne vide ("") est retournée.

STOP

STOP
Ne requiert pas de paramètre

Description

La commande **STOP** est destinée à être utilisée dans une méthode projet de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

Si vous n'avez pas installé de méthode projet de gestion d'erreurs, lorsqu'une erreur se produit (par exemple une erreur de la base de données), 4D affiche sa boîte de dialogue d'erreur standard et interrompt l'exécution de votre code :

- Si le code en cours d'exécution est une méthode objet ou formulaire (ou une méthode projet appelée depuis une méthode formulaire ou objet), 4D "rend la main" au formulaire actuellement affiché.
- Si le code en cours d'exécution est une méthode appelée depuis un menu, 4D "rend la main" à la barre de menus ou au formulaire actuellement affiché.
- Si le code en cours d'exécution est la méthode de gestion d'un process, le process est tué.
- Si le code en cours d'exécution est une méthode appelée directement ou indirectement par une opération d'import ou d'export, cette dernière est stoppée. Il en va de même pour les recherches séquentielles et les tris.
- Etc...

Si vous décidez de traiter les erreurs à l'aide d'une méthode projet d'interception d'erreurs, 4D n'affiche plus sa boîte de dialogue d'erreur standard et n'interrompt plus l'exécution de votre code. Au lieu de cela, 4D appelle votre méthode projet d'interception d'erreurs puis poursuit l'exécution de la ligne de code suivant celle ayant provoqué l'erreur. Vous pouvez traiter certaines erreurs par programmation (par exemple pendant un import, si vous interceptez une erreur de la base de donnée signalant une valeur dupliquée, vous pouvez ignorer l'erreur et poursuivre l'opération). Il existe également des erreurs que vous ne pouvez pas traiter ou des erreurs que vous ne devez pas "ignorer". Dans ces cas, vous devez stopper l'exécution de la méthode comme le fait 4D ; pour cela, appelez la commande **STOP** depuis la méthode projet d'interception d'erreurs.

Note historique

Bien que la commande **STOP** soit destinée à une utilisation au sein d'une méthode projet d'interception d'erreurs, des membres de la communauté 4D ont commencé à l'utiliser dans d'autres méthodes projet pour interrompre leur exécution. Le fait que cela fonctionne n'est qu'un "effet secondaire". Nous vous recommandons de n'utiliser cette commande que dans des méthodes projet d'interception d'erreurs.

JSON

Présentation des commandes JSON

Les commandes JSON permettent de générer et de *parser* (analyser) des objets de langage au format JSON. Le format JSON rend notamment possible l'accès aux bases 4D (structure et données) via un navigateur Web.

La prise en charge d'objets structurés est une nouveauté majeure du langage de 4D v14, destinée à faciliter l'échange de données structurées. Les commandes du thème "JSON" permettent à 4D de manipuler directement des objets JSON. Toutefois, le programme peut également manipuler des objets "natifs" (dont la structure est inspirée du JSON), permettant des échanges vers tout type de langage. Pour plus d'informations, reportez-vous au chapitre **Objets (Langage)**.

Introduction à JSON

"JSON (*JavaScript Object Notation*) est un format de données textuelles, générique, dérivé de la notation des objets du langage ECMAScript." (*source : Wikipédia*). JSON est un format indépendant de tout autre langage, mais utilise des conventions qui sont familières aux développeurs C++ ou JavaScript, Perl, etc. JSON est un format particulièrement adapté à l'échange de données.

Ce paragraphe résume les principes de notation mis en oeuvre dans JSON. Pour une description complète de ce format, veuillez vous reporter au site www.json.org/json-fr.html.

Syntaxe JSON

La syntaxe JSON est basée sur les principes suivants :

- les données sont des paires nom/valeur,
- les données sont séparées par des virgules,
- les objets sont définis par des accolades {},
- les tableaux sont définis par des crochets [ ].

Propriétés JSON

Les données JSON sont exprimées sous forme de paires nom/valeur (ou clé/valeur). Une paire nom/valeur contient un nom de champ (entre guillemets), suivi de deux-points, suivi d'une valeur. Par exemple :

```
"firstName":"John"
```

Pour information, cet exemple équivaut en JavaScript à :

```
firstName="John"
```

Attention, les noms de propriétés sont diacritiques et tiennent compte de la casse. Si vous écrivez "FirstName" au lieu de "firstName", vous obtenez une nouvelle paire name/valeur.

Types de données JSON

Les types de valeurs suivants sont pris en charge dans JSON :

Type	Description	Commentaire
chaîne	Tout caractère Unicode excepté " et \ Les valeurs, comme les noms de propriétés, sont encadrées de ", par exemple "ville":"Paris"	\ est utilisé pour les caractères de contrôle : \" = guillemets \\ = barre oblique inversée \/ = barre oblique \b = retour arrière \f = formfeed \n = retour ligne \r = retour chariot \t = tabulation \u = quatre chiffres hexadécimaux
nombre	Entier ou nombre à virgule flottante	Nombres semblables au C ou au Java sauf que les formats octal et hexadécimal ne sont pas utilisés
objet	{ }	
tableau	[ ]	
booléen	true ou false	
null	null	

Objets JSON

Les objets JSON sont définis par des accolades. Ils peuvent contenir un nombre indéfini de paires noms/valeurs, par exemple :

```
{ "firstName":"John" , "lastName":"Doe" }
```

Tableaux JSON

Les tableaux JSON sont définis par des crochets. Un tableau peut contenir un nombre indéfini d'objets :

```
{     "employees": [         { "firstName":"John" , "lastName":"Doe" },         { "firstName":"Anna" , "lastName":"Smith" },         {
"firstName":"Peter" , "lastName":"Jones" }     ] }
```

Prise en charge du fuseau horaire

Les conversions des dates 4D vers et depuis JSON tiennent compte par défaut du fuseau horaire (*timezone*) de la machine sur laquelle elles ont eu lieu (conformité JavaScript). Par exemple, en France (GMT+2), la conversion de !23/08/2013! donne "2013-08-22T22:00:00Z" et inversement.

Vous pouvez modifier ce fonctionnement et ne pas tenir compte du fuseau horaire, lors de la mise en place de procédures d'exportation par exemple, à

l'aide de la commande **FIXER PARAMETRE BASE**.
Pour plus d'informations sur la conversion des dates 4D/JSON, reportez-vous au paragraphe **Conversion des dates JavaScript**.

JSON Parse

JSON Parse (chaîneJSON {; type}) -> Résultat

Paramètre	Type		Description
chaîneJSON	Chaîne	➡	Chaîne en JSON à analyser
type	Entier long	➡	Type dans lequel convertir les valeurs
Résultat	Booléen, Objet, Pointeur, Réel, Texte	⬅	Valeurs extraites de la chaîne JSON

Description

La commande **JSON Parse** analyse (parse) le contenu d'une chaîne formatée en JSON et en extrait des valeurs que vous pouvez stocker dans un champ ou une variable 4D. Cette commande désérialise des données JSON ; elle effectue l'action inverse de la commande **JSON Stringify**.

Passez dans *chaîneJSON* la chaîne au format JSON dont vous souhaitez analyser le contenu. Cette chaîne doit être correctement formatée, sinon une erreur de parsing est générée. **JSON Parse** peut donc être utilisée pour valider du JSON.

Note : Si vous utilisez des pointeurs, vous devez appeler la commande **JSON Stringify** avant **JSON Parse**.

Par défaut, si vous omettez le paramètre *type*, 4D tentera de convertir la valeur obtenue dans le type de la variable ou du champ utilisé pour stocker le résultat (s'il est défini). Sinon, 4D tentera de déduire le type. Vous pouvez également forcer l'interprétation du type en passant le paramètre *type* : passez une des constantes suivantes du thème **Types champs et variables** :

Constante	Type	Valeur
Est un booléen	Entier long	6
Est un entier long	Entier long	9
Est un numérique	Entier long	1
Est un objet	Entier long	38
Est un texte	Entier long	2
Est une date	Entier long	4

Notes :

- Les valeurs de type numérique doivent être incluses dans l'intervalle ±10.421e±10
- Dans les valeurs de type texte, tous les caractères spéciaux doivent être échappés, y compris les guillemets (cf. exemples)
- Les date JSON doivent être au format "\"YYYY-MM-DDTHH:mm:ssZ\"". La commande considère que la date 4D contient une heure locale et non GMT.

Exemple 1

Exemples de conversions simples :

```
C_REEL($r)
$r:=JSON Parse("42.17") //$r = 42,17 (réel)

C_ENTIER LONG($el)
$el:=JSON Parse("120.13";Est un entier long) //$el=120

C_TEXTE($t)
$t:=JSON Parse("\"Année 42\"";Est un texte) //$t="Année 42" (texte)

C_OBJET($o)
$o:=JSON Parse("{\"name\":\"jean\"}")
    // $o = {"name":"jean"} (objet 4D)

C_BOOLEEN($b)
$b:=JSON Parse("{\"manager\":true}";Est un booléen) //$b=vrai
```

Exemple 2

Exemples de conversions de données de type date :

```
$test:=JSON Parse("\"1990-12-25T12:00:00Z\"")
    // $test=1990-12-25T12:00:00Z
C_DATE($date)
$date:=JSON Parse("\"2008-01-01T12:00:00Z\"";Est une date)
    //$date=01/01/08
```

Exemple 3

Cet exemple montre l'utilisation conjointe des commandes **JSON Stringify** et **JSON Parse** :

```
C_TEXTE($MyContact)
C_OBJET($Contact)

 // JSON Stringify : conversion d'un objet JSON en chaîne JSON
```

```
$MyContact:=JSON Stringify("{\"name\":\"Monroe\",\"firstname\":\"Alan\"}")
 // $MyContact = "{\\"name\\":\\"Monroe\\",\\"firstname\\":\\"Alan\\"}"
 // JSON Parse : conversion d'une chaîne JSON en objet JSON
$Contact:=JSON Parse("{\"name\":\"Monroe\",\"firstname\":\"Alan\"}")
 // $Contact = {"name":"Monroe","firstname":"Alan"}
```

JSON PARSE TABLEAU

JSON PARSE TABLEAU (chaîneJSON ; tabObjet)

Paramètre	Type		Description
chaîneJSON	Chaîne	⇨	Chaîne en JSON à analyser
tabObjet	Tableau objet, Tableau texte, Tableau réel, Tableau booléen, Tableau pointeur	⇦	Tableau contenant le résultat de l'analyse de la chaîne JSON

Description

La commande **JSON PARSE TABLEAU** analyse (*parse*) le contenu d'une chaîne formatée en JSON et place les données extraites dans le tableau *tabObjet*. Cette commande désérialise des données JSON ; elle effectue l'action inverse de la commande **JSON Stringify tableau**.

Passez dans *chaîneJSON* la chaîne au format JSON dont vous souhaitez analyser le contenu. Cette chaîne doit être correctement formatée, sinon une erreur d'analyse est générée.

Passez dans *tabObjet* le tableau objet devant recevoir le résultat de l'analyse.

Exemple

Dans cet exemple, les données des champs des enregistrements d'une table sont extraites puis placées dans des tableaux d'objets :

```
C_OBJET($ref)
TABLEAU OBJET($sel;0)
TABLEAU OBJET($sel2;0)
C_TEXTE(v_String)

OB FIXER($ref;"name";->[Company]Company Name)
OB FIXER($ref;"city";->[Company]City)

Tant que(Non(Fin de selection([Company])))
      $ref_company:=OB Copier($ref;Vrai)
      AJOUTER A TABLEAU($sel;$ref_company)
            // $sel{1}={"name":"4D SAS","city":"Clichy"}
            // $sel{2}={"name":"MyComp","city":"Lyon"}
            // ...
      ENREGISTREMENT SUIVANT([Company])
Fin tant que

v_String:=JSON Stringify tableau($sel)
    // v_String= [{"name":"4D SAS","city":"Clichy"},{"name":"MyComp","city":"Lyon"}...]
JSON PARSE TABLEAU(v_String;$sel2)
          // $sel2{1}={"name":"4D SAS","city":"Clichy"}
          // $sel2{2}={"name":"MyComp","city":"Lyon"}
          //...
```

JSON Stringify

JSON Stringify (valeur {; *}) -> Résultat

Paramètre	Type		Description
valeur	Objet, Tableau objet, Chaîne, Réel, Date, Heure	➡	Données à convertir en chaîne JSON
*	Opérateur	➡	Améliorer la présentation
Résultat	Texte	⮌	Chaîne contenant le texte JSON sérialisé

Description

La commande **JSON Stringify** convertit le paramètre *valeur* en une chaîne JSON. Cette commande sérialise des données en JSON ; elle effectue l'action inverse de la commande **JSON Parse**.

Passez dans *valeur* les données à sérialiser. Elles peuvent être exprimées sous forme scalaire (chaîne, numérique, date ou heure) ou via un objet 4D (ou un tableau d'objets).

Dans le cas d'un objet, vous pouvez inclure tout type de valeurs (cf. paragraphe **Types de données JSON**). Le formatage JSON doit respecter les règles suivantes :

- les valeurs de type chaîne doivent être encadrées de guillemets. Tous les caractères Unicode peuvent être utilisés à l'exception des caractères spéciaux, devant être précédés par une barre oblique inversée.
- numérique : intervalle ±10.421e±10
- booléen : chaîne "true" ou "false"
- pointeur vers un champ, une variable ou un tableau (le pointeur est évalué au moment du stringify)
- date : type texte
- heure : type réel

Vous pouvez passer le paramètre optionnel * afin d'inclure des caractères de formatage dans la chaîne résultante. Cette option permet d'améliorer la présentation des données JSON (*pretty formatting*).

Exemple 1

Conversions de valeurs scalaires :

```
$vc:=JSON Stringify("Saperlipopette") // "Saperlipopette"
$vel:=JSON Stringify(120) // "120"
$vd:=JSON Stringify(!28/08/2013!) //  "2013-08-27T22:00:00Z"
$vh:=JSON Stringify(?20:00:00?) // "72000000" secondes depuis minuit
```

Exemple 2

Conversion d'une chaîne contenant des caractères spéciaux :

```
$s:=JSON Stringify("{\"name\":\"john\"}")
    // $s="{\\"name\\":\\"john\\"}"
$p:=JSON Parse($s)
    // $p={"name":"john"}
```

Exemple 3

Exemple utilisant un pointeur vers une variable :

```
C_OBJET($MaVarTest)
C_TEXTE($name ;$jsonstring )
OB FIXER($MaVarTest;"name";->$name) // définition de l'objet
    // $MaVarTest = {"name":"->$name"}

$jsonstring :=JSON Stringify($MaVarTest)
    // $jsonstring ="{"name":""}"
    //...

$name:="Smith"
$jsonstring :=JSON Stringify($MaVarTest)
    //$jsonstring = "{"name" : "Smith"}"
```

Exemple 4

Sérialisation d'un objet 4D :

```
C_TEXTE($varjsonTextserialized)
C_OBJET($Contact)
OB FIXER($Contact;"firstname";"Alan")
```

```
OB FIXER($Contact;"lastname";"Monroe")
OB FIXER($Contact;"age";40)
OB FIXER($Contact;"phone";"[555-0100,555-0120]")

$varjsonTextserialized:=JSON Stringify($Contact)

    // $varjsonTextserialized = "{"lastname":"Monroe","phone":"[555-0100,
    // 555-0120]","age":40,"firstname":"Alan"}"
```

Exemple 5

Exemples de sérialisation d'un objet 4D avec et sans le paramètre *:

```
C_TEXTE($MyContact)
C_TEXTE($MyPContact)
C_OBJET($Contact;$Children)
OB FIXER($Contact;"lastname";"Monroe";"firstname";"Alan")
OB FIXER($Children;"firstname";"Jim";"age";"12")
OB FIXER($Contact;"children";$Children)
$MyContact:=JSON Stringify($Contact)
$MyPContact:=JSON Stringify($Contact;*)
    //$MyContact= {"lastname":"Monroe","firstname":"Alan","children":{"firstname":"John","age":"12"}}
    //$MyPContact= {\n\t"lastname": "Monroe",\n\t"firstname": "Alan",\n\t"children": {\n\t\t"firstname":
"John",\n\t\t"age": "12"\n\t}\n}
```

L'intérêt de ce formatage apparaît clairement lorsque le JSON est représenté dans une zone Web :

- Formatage standard :

```
{"Name":"Monroe","firstname":"Alan","children":{"firstname":"John","age":12}}
```

- Formatage amélioré :

```
{
        "Name": "Monroe",
        "firstname": "Alan",
        "children": {
                "firstname": "John",
                "age": 12
        }
}
```

JSON Stringify tableau

JSON Stringify tableau (tab {; *}) -> Résultat

Paramètre	Type		Description
tab	Tableau texte, Tableau réel, Tableau booléen, Tableau pointeur, Tableau objet	➡	Tableau dont le contenu doit être sérialisé
*	Opérateur	➡	Améliorer le formatage
Résultat	Texte	⬅	Chaîne contenant le tableau JSON sérialisé

Description

La commande **JSON Stringify tableau** convertit le tableau 4D *tab* en un tableau JSON sérialisé. Cette commande effectue l'action inverse de la commande **JSON PARSE TABLEAU**.

Passez dans *tab* un tableau 4D contenant les données à sérialiser. Le tableau peut être de type texte, réel, booléen, pointeur ou objet.

Vous pouvez passer le paramètre optionnel * afin d'inclure des caractères de formatage dans la chaîne résultante. Cette option permet d'améliorer la présentation des données JSON lorsqu'elles sont affichées dans une page Web (*pretty formatting*).

Exemple 1

Conversion d'un tableau texte :

```
C_TEXTE($jsonString)
TABLEAU TEXTE($ArrayFirstname;2)
$ArrayFirstname{1}:="John"
$ArrayFirstname{2}:="Jim"
$jsonString :=JSON Stringify tableau($ArrayFirstname)

    // $jsonString = "["John","Jim"]"
```

Exemple 2

Conversion d'un tableau texte contenant des nombres :

```
TABLEAU TEXTE($phoneNumbers;0)
AJOUTER A TABLEAU($phoneNumbers ;"555-0100")
AJOUTER A TABLEAU($phoneNumbers ;"555-0120")
$string :=JSON Stringify tableau($phoneNumbers)
    // $string = "["555-0100","555-0120"]"
```

Exemple 3

Conversion d'un tableau objet :

```
C_OBJET($ref_john)
C_OBJET($ref_jim)
TABLEAU OBJET($myArray;0)
OB FIXER($ref_john;"name";"John";"age";35)
OB FIXER($ref_jim;"name";"Jim";"age";40)
AJOUTER A TABLEAU($myArray ;$ref_john)
AJOUTER A TABLEAU($myArray ;$ref_jim)
$JsonString :=JSON Stringify tableau($myArray)
    // $JsonString = "[{"name":"John","age":35},{"name":"Jim","age":40}]"

    // Si vous souhaitez visualiser le résultat dans une page Web, passez
    // le paramètre optionnel * :
$JsonStringPretty :=JSON Stringify tableau($myArray;*)
```

```
[
  {
    "name": "John",
    "age": 35
  },
  {
    "name": "Jim",
    "age": 40
  }
]
```

Exemple 4

Conversion d'une sélection 4D dans un tableau objet :

```
C_OBJET($jsonObject)
```

```
C_TEXTE($jsonString)

CHERCHER([Company];[Company]Company Name="a@")
OB FIXER($jsonObject;"company name";->[Company]Company Name)
OB FIXER($jsonObject;"city";->[Company]City)
OB FIXER($jsonObject;"date";[Company]Date_input)
OB FIXER($jsonObject;"time";[Company]Time_input)
TABLEAU OBJET($arraySel;0)

Tant que(Non(Fin de selection([Company])))
      $ref_value:=OB Copier($jsonObject;Vrai)
          // Si vous ne les copiez pas, les valeurs seront des chaînes vides
      AJOUTER A TABLEAU($arraySel;$ref_value)
          // Chaque élément contient les valeurs sélectionnées, par exemple :
          // $arraySel{1} = // {"company name":"APPLE","time":43200000,"city":
          // "Paris","date":"2012-08-02T00:00:00Z"}
      ENREGISTREMENT SUIVANT([Company])
Fin tant que

$jsonString:=JSON Stringify tableau($arraySel)
    // $jsonString = "[{"company name":"APPLE","time":43200000,"city":
    //"Paris","date":"2012-08-02T00:00:00Z"},{"company name":
    //"ALMANZA",...}]"
```

JSON VERS SELECTION

JSON VERS SELECTION (laTable ; jsonTab)

Paramètre	Type		Description
laTable	Table	➡	Table 4D dans laquelle copier les éléments
jsonTab	Texte	➡	Tableau d'objets en JSON

Description

La commande **JSON VERS SELECTION** copie le contenu du tableau d'objets JSON *jsonTab* vers la sélection d'enregistrements de *laTable*.

Le paramètre *jsonTab* est un *texte* représentant un tableau d'objets JSON contenant un ou plusieurs élément(s). Le format attendu est du type :

```
"[{"attribut1":"valeur1","attribut2":"valeur2",...},...,{"attribut1":"valeurN","attribut2":"valeurN",...}]"
```

Si une sélection existe pour *laTable* au moment de l'appel, les éléments du tableau JSON sont copiés dans les enregistrements en fonction de l'ordre du tableau et de l'ordre des enregistrements. Si le nombre d'éléments définis dans le tableau JSON est supérieur au nombre d'enregistrements de la sélection courante, de nouveaux enregistrements sont créés. Les enregistrements, qu'ils soient nouveaux ou existants, sont automatiquement sauvegardés.

Note : Cette commande prend en charge les champs de type objet : les données JSON sont automatiquement converties.

Attention : Comme **JSON VERS SELECTION** remplace les informations éventuellement présentes dans les enregistrements existants, cette commande doit être utilisée avec prudence.

Si un enregistrement est verrouillé par un autre process pendant l'exécution de la commande, il n'est pas modifié. Tous les enregistrements verrouillés sont placés dans l'**Ensemble système LockedSet**. Après l'exécution de **JSON VERS SELECTION**, vous pouvez tester si l'ensemble *LockedSet* contient des enregistrements qui étaient verrouillés.

Exemple

Utilisation de la commande **JSON VERS SELECTION** pour ajouter des enregistrements dans la table [Company] :

```
C_OBJET($Object1;$Object2;$Object3;$Object4)
C_TEXTE($ObjectString)
TABLEAU OBJET($arrayObject;0)

OB FIXER($Object1;"ID";"200";"Company Name";"4D SAS";"City";"Clichy")
AJOUTER A TABLEAU($arrayObject;$Object1)

OB FIXER($Object2;"ID";"201";"Company Name";"APPLE";"City";"Paris")
AJOUTER A TABLEAU($arrayObject;$Object2)

OB FIXER($Object3;"ID";"202";"Company Name";"IBM";"City";"London")
AJOUTER A TABLEAU($arrayObject;$Object3)

OB FIXER($Object4;"ID";"203";"Company Name";"MICROSOFT";"City";"New York")
AJOUTER A TABLEAU($arrayObject;$Object4)

$ObjectString:=JSON Stringify tableau($arrayObject)

    // $ObjectString = "[{"ID":"200","City":"Clichy","Company Name":"4D
    // SAS"},{"ID":"201","City":"Paris","Company Name":"APPLE"},{"ID":"202",
    //"City":"London","Company Name":"IBM"},{"ID":"203","City":"New
    //York","Company Name":"MICROSOFT"}]"

JSON VERS SELECTION([Company];$ObjectString)
    // vous créez 4 enregistrements dans la table [Company], remplissant les
    //champs ID, Company name et city
```

Selection vers JSON

Selection vers JSON (laTable {; leChamp}{; leChamp2 ; ... ; leChampN}{; template}) -> Résultat

Paramètre	Type		Description
laTable	Table	➡	Table à sérialiser
leChamp	Champ	➡	Champ(s) dont le contenu doit être sérialisé
template	Objet	➡	Objet pour la sélection de libellés et de champs
Résultat	Texte	⬅	Chaîne contenant le tableau JSON sérialisé

Description

La commande **Selection vers JSON** retourne une chaîne qui contient un tableau JSON avec autant d'éléments qu'il y a d'enregistrements dans la sélection courante de *laTable*. Chaque élément du tableau est un objet JSON contenant les libellés et les valeurs des champs de la sélection.

Si vous passez uniquement le paramètre *laTable*, la commande inclut dans le tableau JSON les valeurs de tous les champs de la table exprimables en JSON. Les champs de type BLOB et image sont ignorés.

Si vous ne souhaitez pas inclure tous les champs de *laTable*, vous pouvez utiliser soit le paramètre *leChamp* soit le paramètre *template* :

- *leChamp* : passez un ou plusieurs champ(s) dans ce paramètre. Seules les valeurs des champs définis seront incluses dans le tableau JSON.
- *template* : passez un objet 4D contenant une ou plusieurs paire(s) *nom/valeur* où *nom* peut être tout nom d'attribut valide et *valeur* est un pointeur vers un champ à inclure. Cette syntaxe permet de personnaliser les libellés des champs dans le tableau JSON.

Cette commande prend en charge les champs de type objet : les données des champs sont automatiquement converties au format JSON. A noter que l'instruction 4D suivante sera interprétée comme "produire du JSON à partir de toutes les valeurs de *champObjet* dans la sélection courante de la table" :

```
Selection vers JSON([uneTable];champObjet)
```

Exemple 1

Vous voulez créer une chaîne JSON représentant cette sélection :

newbase - Adhérents : 6 enregistrement(s) sur 6

Nom :	Prénom :	Adresse :	Code postal :	Ville :
Durant	Marc	25 rue du parc	95000	Pontoise
Smith	John	24, rue Philibert-Delorme	75017	Paris
Auquart	Adémart	37, quai de l'Iton	37100	Tours
Aubert	Jean-Marc	48bis, boulevard du Montparnas	75014	Paris
Lenuze	Roland	184, chaussée de Tournai	59000	Lille
Pradel	Jacques	68, impasse Notre-Dame	06120	Antibes

1) Vous souhaitez inclure les valeurs de tous les champs de la table [Adhérents] :

```
$jsonString :=Selection vers JSON([Adhérents])
    // $jsonString = [{"Nom":"Durant","Prénom":"Marc","Adresse":"25 rue du
    //parc","Code postal":"95000","Ville":"Pontoise"},{"Nom":"Smith",
    //"Prénom":"John","Adresse":"24, rue Philibert-Delorme ","Code postal":
    //"75017","Ville":"Paris"},{"Nom":"Auquart","Prénom":"Adémart",
    //"Adresse":"37, quai de l´Iton","Code postal":"37100","Ville":"Tours"},...]
```

2) Vous souhaitez réduire la sélection et n'inclure que deux champs dans la chaîne JSON en utilisant la syntaxe basée sur les champs :

```
CHERCHER([Adhérents];[Adhérents]Nom="A@")
$jsonString :=Selection vers JSON([Adhérents];[Adhérents]Nom;[Adhérents]Ville)
    // $jsonString = [{"Nom":"Auquart","Ville":"Tours"},{"Nom":"Aubert","Ville":"Paris"}]
```

3) Vous souhaitez n'inclure qu'un champ dans la chaîne JSON et utiliser un autre libellé. Vous utilisez la syntaxe *template* :

```
C_OBJET($template)
OB FIXER($template;"Membre";->[Adhérents]Nom) //libellé personnalisé et un seul champ
TOUT SELECTIONNER([Adhérents])
$jsonString :=Selection vers JSON([Adhérents];$template)
    // $jsonString = [{"Membre":"Durant"},{"Membre":"Smith"},{"Membre":"Auquart"},{"Membre":"Aubert"},
{"Membre":"Lenuze"},{"Membre":"Pradel"}]
```

Exemple 2

Vous utilisez la syntaxe avec *template* afin d'exporter des champs de différentes tables :

```
C_OBJET($template)
C_TEXTE($chaineJSON)
OB FIXER($template;"Nom";->[Emp]Nom)
OB FIXER($template;"Prénom";->[Emp]Prénom)
OB FIXER($template;"Société";->[Société]Nom) //libellé personnalisé sinon conflit avec le champ [Emp]Nom
TOUT SELECTIONNER([Emp])
```

```
FIXER LIEN CHAMP([Emp]UUID_Societe;Automatique;Ne pas changer)
$chaineJSON:=Selection vers JSON([Emp];$template)
FIXER LIEN CHAMP([Emp]UUID_Societe;Configuration structure;Ne pas changer)
```

Langage

- Est une variable
- EXECUTER METHODE
- EXECUTER METHODE DANS SOUS FORMULAIRE
- Nil
- Nom commande
- Nom methode courante
- Nombre de paramètres
- PAS DE TRACE
- Pointeur vers
- RESOUDRE POINTEUR
- Self
- TRACE
- Type

Est une variable

Est une variable (pointeur) -> Résultat

Paramètre	Type		Description
pointeur	Pointeur	→	Pointeur à tester
Résultat	Booléen	←	VRAI = Pointeur pointe vers une variable FAUX = Pointeur ne pointe pas vers une variable

Description

La fonction **Est une variable** retourne Vrai si le pointeur passé dans le paramètre *pointeur* référence une variable définie. Elle retourne Faux dans tous les autres cas (pointeur vers un champ ou table, pointeur Nil, etc.).

Si vous souhaitez connaître le nom de la variable pointée ou le numéro du champ, vous pouvez utiliser la commande **RESOUDRE POINTEUR**.

EXECUTER METHODE

EXECUTER METHODE (nomMéthode {; résultat {; param}}{; param2 ; ... ; paramN})

Paramètre	Type		Description
nomMéthode	Chaîne	⇒	Nom de méthode projet à exécuter
résultat	Variable, Opérateur	⇐	Variable recevant le résultat de la méthode ou * pour une méthode ne retournant pas de résultat
param	Expression	⇒	Paramètre(s) de la méthode

Description

La commande **EXECUTER METHODE** provoque l'exécution de la méthode projet *nomMéthode* en lui passant éventuellement les paramètres *param1...paramN*. Vous pouvez passer tout nom de méthode appelable depuis la base ou le composant exécutant la commande.

Passez dans *résultat* une variable devant recevoir le résultat de l'exécution de *nomMéthode* (valeur placée dans $0 à l'intérieur de *nomMéthode*). Si la méthode ne retourne pas de résultat, passez * comme deuxième paramètre. Si la méthode ne retourne pas de résultat et ne nécessite pas non plus le passage de paramètre(s), passez uniquement le paramètre *nomMéthode*.

Le contexte d'exécution est préservé dans la méthode appelée, ce qui signifie que le formulaire courant et l'éventuel événement formulaire courant restent définis.

Si vous appelez cette commande depuis un composant et passez dans *nomMéthode* un nom de méthode appartenant à la base hôte (ou inversement), la méthode doit avoir été partagée (option "Partager entre composants et base hôte" dans les propriétés de la méthode).

Variables et ensembles système

Si cette commande est exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

EXECUTER METHODE DANS SOUS FORMULAIRE

EXECUTER METHODE DANS SOUS FORMULAIRE (objetSousForm ; nomMéthode {; retour {; param} {; param2 ; ... ; paramN}})

Paramètre	Type		Description
objetSousForm	Texte	➡	Nom de l'objet sous-formulaire
nomMéthode	Texte	➡	Nom de la méthode projet à exécuter
retour	Opérateur, Variable	➡	* si la méthode ne retourne pas de valeur
		⬅	Valeur retournée par la méthode
param	Variable	➡	Paramètre(s) à passer à la méthode

Description

La commande **EXECUTER METHODE DANS SOUS FORMULAIRE** permet d'exécuter la méthode projet *nomMéthode* dans le contexte de l'objet de sous-formulaire *objetSousForm*.

La méthode projet appelée peut recevoir de 1 à N paramètres dans *param* et retourner une valeur dans *retour*. Passez * dans le paramètre *retour* si la méthode ne retourne pas de paramètres.

Vous pouvez passer dans *nomMéthode* le nom de toute méthode projet accessible depuis la base ou le composant exécutant la commande. Le contexte d'exécution est préservé dans la méthode appelée, ce qui signifie que le formulaire courant et l'événement formulaire courant restent définis. Si le sous-formulaire provient d'un composant, la méthode doit appartenir au composant et disposer de la propriété "Partagée entre les composants et la base hôte".

Cette commande doit être appelée dans le contexte du formulaire parent (contenant l'objet *objetSousForm*), par exemple via sa méthode formulaire.

Note : La méthode *nomMéthode* n'est pas exécutée si *objetSousForm* ne se trouve pas dans la page courante ou n'est pas encore instancié.

Exemple 1

Soit le formulaire "ContactDétail" utilisé comme sous-formulaire dans le formulaire parent "Société". L'objet sous-formulaire qui contient le formulaire ContactDétail est nommé "ContactSousForm". Imaginons que nous souhaitions modifier l'apparence de certains éléments du sous-formulaire en fonction de la valeur de champ(s) de la société (par exemple, "nomcontact" doit passer en rouge lorsque [Société]Ville="New York" et en bleu lorsque [Société]Ville="San Diego"). Ce mécanisme est mis en oeuvre via la méthode ***SetToColor***. Pour pouvoir obtenir ce résultat, la méthode ***SetToColor*** ne peut pas être appelée directement depuis le process de l'événement formulaire "Sur chargement" du formulaire parent Société car l'objet "nomcontact" n'appartient pas au formulaire courant, mais au formulaire affiché dans l'objet sous-formulaire "ContactSousForm". La méthode doit donc être exécutée à l'aide de **EXECUTER METHODE DANS SOUS FORMULAIRE** pour pouvoir fonctionner correctement.

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      Au cas ou
         :([Société]Ville ="New York")
            $Color:=$Red
         :([Société]Ville ="San Diego")
            $Color:=$Blue
         Sinon
            $Color:=$Black
      Fin de cas
      EXECUTER METHODE DANS SOUS FORMULAIRE("ContactSousForm";"SetToColor";*;$Color)
Fin de cas
```

Exemple 2

Vous développez une base de données qui sera utilisée comme composant. Elle comporte un formulaire projet partagé (nommé par exemple Calendrier) contenant des *variables dynamiques* ainsi qu'une méthode projet publique permettant de régler le calendrier : ***SetCalendarDate(varDate)***.
Si cette méthode était utilisée directement dans la méthode du formulaire Calendrier, vous pourriez l'appeler directement dans l'événement "Sur chargement" :

```
SetCalendarDate(Date du jour)
```

Mais, dans le contexte de la base hôte, imaginons qu'un formulaire projet contienne deux sous-formulaires "Calendrier", dans des objets sous-formulaire appelés "Cal1" et "Cal2". Vous devez régler la date de Cal1 au 01/01/10 et celle de Cal2 au 05/05/10. Vous ne pouvez pas appeler directement ***SetCalendarDate*** car la méthode ne "saura" pas à quels formulaire et variables elle devra s'appliquer. Vous devez donc l'appeler via le code suivant :

```
EXECUTER METHODE DANS SOUS FORMULAIRE("Cal1";"SetCalendarDate";*;!01/01/10!)
EXECUTER METHODE DANS SOUS FORMULAIRE("Cal2";"SetCalendarDate";*;!05/05/10!)
```

Exemple 3

Exemple avancé : dans le même contexte que précédemment, cet exemple propose une méthode générique :

```
 // Contenu de la méthode SetCalendarDate
C_DATE($1)
C_TEXTE($2)
Au cas ou
   :(Nombre de paramètres=1)
 // Exécution standard de la méthode (comme si elle était exécutée depuis le formulaire lui-même) ou
```

```
    // spécifiquement pour un contexte (voir cas 2)

      :(Nombre de paramètres=2)
    // Appel depuis l'extérieur, a besoin d'un contexte
    // Appel récursif avec un seul paramètre
         EXECUTER METHODE DANS SOUS FORMULAIRE($2;"SetCalendarDate";*;$1)
  Fin de cas
```

Variables et ensembles système

Si cette commande est exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Nil

Nil (pointeur) -> Résultat

Paramètre	Type		Description
pointeur	Pointeur	⇒	Pointeur à tester
Résultat	Booléen	⇐	VRAI = Pointeur Nil (->[]) FAUX = Pointeur valide vers un objet existant

Description

Nil retourne Vrai si le pointeur que vous passez dans *pointeur* est Nil (->[]). Elle retourne Faux dans tous les autres cas (pointeur vers un champ, une table ou une variable).

Si vous souhaitez connaître le nom de la variable pointée ou le numéro du champ, vous pouvez utiliser la commande **RESOUDRE POINTEUR**.

Nom commande

Nom commande (commande {; info {; thème}}) -> Résultat

Paramètre	Type		Description
commande	Entier long	➡	Numéro de la commande
info	Entier long	⬅	Propriété thread-safe de la commande
thème	Texte	⬅	Thème du langage de la commande
Résultat	Chaîne	➲	Nom de la commande

Description

La fonction **Nom commande** retourne le nom ainsi que (optionnellement) les propriétés de la commande dont le numéro a été passé dans *commande*.

Note : Le numéro de chaque commande est indiqué dans l'Explorateur ainsi que dans la zone Propriétés de cette documentation.

Note de compatibilité : Le nom d'une commande pouvant varier au fil des versions de 4D (commandes renommées) ou en fonction de la langue de l'application, cette commande était utilisée dans les versions précédentes du programme pour désigner une commande directement via son numéro, en particulier dans les parties de code non tokenisées. Ce besoin a diminué au fil des évolutions de 4D, car pour les instructions non tokenisées (formules), 4D propose désormais une *syntaxe tokenisée* permettant de s'affranchir des variations des noms de commandes mais aussi des autres éléments comme les tables, tout en permettant de les saisir de façon lisible (pour plus d'informations sur ce point, reportez-vous à la section **Utiliser des tokens dans les formules**). Par ailleurs, par défaut la version anglaise du langage est utilisée à compter de 4D v15 (toutefois l'option "Utiliser langage français et paramètres régionaux système" de la **Page Méthodes** des Préférences permet de continuer à utiliser la version française dans un 4D français).

Deux paramètres optionnels sont disponibles :

- *info* : propriétés de la commande. La valeur retournée est un champ de bits, dans lequel pour le moment seul le premier bit est signifiant (bit 0). Il est à 1 si la commande est *thread-safe* (i.e. compatible avec une exécution dans un process préemptif) et à 0 si la commande est *thread-unsafe*. Seules les commandes *thread-safe* peuvent être utilisées dans les process préemptifs. Pour plus d'informations sur ce point, veuillez vous reporter à la section **Process 4D préemptifs**.
- *thème* : retourne le nom du thème de la commande dans le langage 4D.

La commande **Nom commande** met la variable OK à 1 si la commande correspond à un numéro de commande existant, et à 0 dans le cas contraire. A noter cependant que certaines commandes existantes ont été désactivées, auquel cas **Nom commande** retourne une chaîne vide.

Exemple 1

Le code suivant vous permet de charger toutes les commandes 4D valides dans un tableau :

```
C_ENTIER LONG($Lon_id)
C_TEXTE($Txt_command)
TABLEAU ENTIER LONG($tLon_Command_IDs;0)
TABLEAU TEXTE($tTxt_commands;0)

Repeter
   $Lon_id:=$Lon_id+1
   $Txt_command:=Nom commande($Lon_id)
   Si(OK=1) //le numéro de commande existe
      Si(Longueur($Txt_command)>0) //la commande n'est pas désactivée
         AJOUTER A TABLEAU($tTxt_commands;$Txt_command)
         AJOUTER A TABLEAU($tLon_Command_IDs;$Lon_id)
      Fin de si
   Fin de si
Jusque(OK=0) //fin des commandes existantes
```

Exemple 2

Dans un formulaire, vous voulez afficher une liste déroulante contenant les commandes standard de génération d'états. Dans la méthode objet de cette liste déroulante, vous écrivez :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TABLEAU TEXTE(asCommand;4)
      asCommand{1}:=Nom commande(1)
      asCommand{2}:=Nom commande(2)
      asCommand{3}:=Nom commande(3)
      asCommand{4}:=Nom commande(4)
   ...
Fin de cas
```

Dans une version anglaise de 4D, la liste déroulante contiendra : Sum, Average, Min et Max.
Dans une version française* de 4D, la liste déroulante contiendra : Somme, Moyenne, Min et Max.

*avec l'application 4D paramétrée pour utiliser le langage français (cf. note de compatibilité),

Exemple 3

Vous souhaitez créer une méthode qui retourne **Vrai** si la commande dont le numéro passé en paramètre est thread-safe, et **Faux** si elle est thread-unsafe.

```
 //Méthode projet Is_Thread_Safe
 //Is_Thread_Safe(numCom) -> Booléen

C_ENTIER LONG($1;$threadsafe)
C_TEXTE($name)
C_BOOLEEN($0)
$name:=Nom commande($1;$threadsafe;$theme)
Si($threadsafe ?? 0) //si le premier bit est à 1
   $0:=Vrai
Sinon
   $0:=Faux
Fin de si
```

Par exemple, pour la commande "STOCKER ENREGISTREMENT", numéro 53, vous pouvez écrire :

```
$isSafe:=Is_Thread_Safe(53)
 // retourne Vrai
```

Nom methode courante

Nom methode courante -> Résultat

Paramètre	Type		Description
Résultat	Chaîne	➲	Nom de la méthode d'appel

Description

La commande **Nom methode courante** retourne le nom de la méthode dans laquelle elle est appelée. Cette commande est utile dans le cadre du débogage de méthodes génériques.

En fonction du type de méthode d'appel, la chaîne retournée peut prendre l'une des formes suivantes :

Méthode d'appel	Chaîne retournée
Méthode base	NomMéthode
Trigger	Trigger sur [NomTable]
Méthode projet	NomMéthode
Méthode formulaire table	[NomTable].NomFormulaire
Méthode formulaire projet	NomFormulaire
Méthode objet formulaire table	[NomTable].NomFormulaire.NomObjet
Méthode objet formulaire projet	NomFormulaire.NomObjet
Méthode projet de composant	NomMéthode
Méthode formulaire projet de composant	NomFormulaire (NomComposant)
Méthode objet formulaire projet de composant	NomFormulaire (NomComposant).NomObjet (NomComposant)

Cette commande ne doit pas être appelée depuis une formule 4D.

Note : Pour que cette commande fonctionne en mode compilé, il est nécessaire que la base ait été compilée avec l'option **Contrôle d'exécution** (située dans les Propriétés de la base) activée.

Pour désactiver localement le contrôle d'exécution dans une méthode (ou une partie de méthode), vous pouvez utiliser les commentaires spéciaux suivants :

```
`%R- pour désactiver le contrôle d'exécution
`%R+ pour activer le contrôle d'exécution
`%R* pour restituer l'état initial du contrôle d'exécution (défini dans les Propriétés de la base).
```

Nombre de paramètres

Nombre de paramètres -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Nombre de paramètres effectivement passés

Description

Nombre de paramètres retourne le nombre de paramètres passés à une méthode projet.

ATTENTION : Nombre de paramètres n'a d'intérêt que dans une méthode projet appelée par une autre méthode (projet ou non). Si la méthode projet qui appelle **Nombre de paramètres** est associée à une commande de menu, la fonction retourne 0.

Exemple 1

Les méthodes projet de 4D acceptent que des paramètres soient optionnels, à partir de la droite. Par exemple, la méthode *maMéthode(a;b;c;d)* peut accepter les syntaxes suivantes :

```
maMéthode(a;b;c;d) ` Tous les paramètres sont passés
maMéthode(a;b;c) ` Le dernier paramètre n'est pas passé
maMéthode(a;b) ` Les deux derniers paramètres ne sont pas passés
maMéthode(a) ` Seul le premier paramètre est passé
maMéthode ` Aucun paramètre n'est passé
```

Si vous utilisez **Nombre de parametres** dans ***maMéthode***, vous pouvez détecter le nombre de paramètres passés et effectuer des opérations différentes selon ce nombre. L'exemple suivant affiche un texte et peut soit l'insérer dans une zone de 4D Write, soit l'écrire dans un document sur disque :

```
  ` Méthode AJOUTER TEXTE
  ` AJOUTER TEXTE ( Texte { ; Entier long { ; Heure } } )
  ` AJOUTER TEXTE ( Texte { ; zone 4D Write { ; RéfDoc } } )

C_TEXTE($1)
C_HEURE($2)
C_ENTIER LONG($3)

MESSAGE($1)
Si(Nombre de paramètres>=3)
   ENVOYER PAQUET($3;$1)
Sinon
   Si(Nombre de paramètres>=2)
      wr_INSERER TEXTE($2;$1)
   Fin de si
Fin de si
```

Vous pouvez ensuite appeler cette méthode de ces trois façons différentes :

```
AJOUTER TEXTE(vtTexte) ` Afficher seulement le message texte
AJOUTER TEXTE(vtTexte;$wrZone) ` Afficher le message texte et ajouter le texte à $wrZone
AJOUTER TEXTE(vtTexte;0;$vhRéfDoc) ` Afficher le message texte et l'écrire dans $vhRéfDoc
```

Exemple 2

Les méthodes projet de 4D acceptent un nombre variable de paramètres du même type à partir de la droite. Pour déclarer ces paramètres, vous devez utiliser des directives de compilation auxquelles vous passez *${N}* en tant que variable, où **N** spécifie le premier des paramètres. A l'aide de **Nombre de paramètres**, vous pouvez référencer ces paramètres dans une boucle avec la syntaxe d'indirection de paramètre. L'exemple suivant est une fonction qui retourne la valeur maximale reçue en tant que paramètre :

```
  ` Méthode projet Max de
  ` Max de ( Réel { ; Réel2... ; RéelN } ) -> Réel
  ` Max de ( Valeur { ; Valeur2... ; ValeurN } ) -> Valeur maximale

C_REEL($0;${1}) ` Tous les paramètres sont de type REEL ainsi que le résultat de la fonction
$0:=${1}
Boucle($vlParam;2;Nombre de paramètres)
   Si(${$vlParam}>$0)
      $0:=${$vlParam}
   Fin de si
Fin de boucle
```

Vous pouvez alors appeler cette méthode d'une des deux manières suivantes :

```
vrRésultat:=Max de(Enregistrements dans ensemble("Opération A");Enregistrements dans ensemble("Opération B"))
```

ou :

```
vrRésultat:=Max de(r1;r2;r3;r4;r5;r6)
```

PAS DE TRACE

PAS DE TRACE
Ne requiert pas de paramètre

Description

La commande **PAS DE TRACE** est utilisée en phase de développement d'une base de données, pour contrôler l'exécution des méthodes.

PAS DE TRACE désactive le débogueur appelé par la commande **TRACE**, par une erreur ou par l'utilisateur. Utiliser **PAS DE TRACE** équivaut à cliquer sur le bouton **Pas de trace** dans la fenêtre de débogage.

Dans les bases compilées, cette commande est ignorée.

Pointeur vers

Pointeur vers (nomVar) -> Résultat

Paramètre	Type		Description
nomVar	Chaîne	⇒	Nom d'une variable process ou interprocess
Résultat	Pointeur	⇐	Pointeur vers une variable process ou interprocess

Description

Pointeur vers retourne un pointeur vers la variable process ou interprocess dont le nom est passé dans *nomVar*.

Pour récupérer un pointeur vers un champ, utilisez la fonction **Champ**. Pour récupérer un pointeur vers une table, utilisez la fonction **Table**.

Note : Vous pouvez passer à **Pointeur vers** des expressions telles que, par exemple, *tTabNom+"{3}"* ainsi que des éléments de tableau 2D (*tTabNom+"{3}{5}"*).
En revanche, vous ne ne pouvez pas passer d'indices variables (*tTabNom+"{maVar}"*).

Exemple 1

Dans un formulaire, vous construisez une grille de 5 X 10 variables saisissables dont les noms sont v1, v2... v50. Pour initialiser toutes ces variables, vous pouvez écrire :

```
` ...
Boucle($vlVar;1;50)
   $vpVar:=Pointeur vers("v"+Chaine($vlVar))
   $vpVar->:=""
Fin de boucle
```

Exemple 2

Utilisation de pointeurs vers des éléments de tableaux à deux dimensions :

```
$pt:=Pointeur vers("a{1}{2}")
 //$pt=->a{1}{2}
$pt2:=Pointeur vers("atCities"+"{2}{6}")
 //$pt2=->atCities{2}{6}
```

RESOUDRE POINTEUR

RESOUDRE POINTEUR (pointeur ; nomVar ; numTable ; numChamp)

Paramètre	Type		Description
pointeur	Pointeur	⇨	Pointeur duquel récupérer l'objet référencé
nomVar	Chaîne	⇦	Nom de la variable référencée ou chaîne vide
numTable	Entier long	⇦	Numéro de la table ou de l'élément de tableau référencé(e) ou 0 ou -1
numChamp	Entier long	⇦	Numéro du champ ou de l'élément de tableau 2D référencé ou 0 ou -1

Description

RESOUDRE POINTEUR récupère l'information de l'objet référencé par *pointeur* et la retourne dans les paramètres *nomVar*, *numTable* et *numChamp*.
Selon la nature de l'objet référencé par le pointeur, **RESOUDRE POINTEUR** retourne les valeurs suivantes :

Objet référencé	Paramètres		
	nomVar	numTable	numChamp
Aucun (pointeur NIL)	"" (chaîne vide)	0	0
Variable	Nom de la variable	-1	-1
Tableau	Nom du tableau	-1	-1
Elément de tableau	Nom du tableau	numéro de l'élément	-1
Elément de tableau 2D	Nom du tableau 2D	numéro de ligne de l'élément	numéro de colonne de l'élément
Table	"" (chaîne vide)	numéro de la table	0
Champ	"" (chaîne vide)	numéro de la table	numéro du champ

Notes :

- Si la valeur que vous passez dans le paramètre *pointeur* n'est pas de type pointeur, une erreur de syntaxe est générée.
- La commande **RESOUDRE POINTEUR** ne fonctionne pas avec les pointeurs vers des variables locales. En effet, par définition plusieurs variables locales de même nom pouvant exister à différents emplacements, il n'est pas possible pour la commande de connaître la variable à dépointer.

Exemple 1

Dans un formulaire, vous créez un groupe de 100 variables saisissables qui s'appellent v1, v2... v100. Pour cela, vous procédez de la manière suivante :

- Vous créez une variable saisissable que vous appelez v.
- Vous définissez les propriétés de l'objet suivant vos besoins.
- Vous associez la méthode suivante à l'objet :

```
FaireQuelqueChose(Self) ` FaireQuelqueChose est une méthode projet de la base
```

- Vous pouvez alors soit dupliquer la variable autant de fois que nécessaire, soit utiliser la fonctionnalité Tableau sur la grille de l'éditeur de formulaires.
- Dans la méthode ***FaireQuelqueChose***, si vous voulez connaître l'indice de la variable pour laquelle la méthode est appelée, vous écrivez le code suivant :

```
RESOUDRE POINTEUR($1;$vaNomVar;$vlNumTable;$vlNumChamp)
$vlVarNum:=Num(Sous chaine($vaNomVar;2))
```

- En suivant ces étapes, vous avez écrit une fois seulement les méthodes objet pour les 100 variables : vous n'avez pas eu besoin d'écrire **FaireQuelqueChose(1), FaireQuelqueChose(2)..., FaireQuelqueChose(100))**.

Exemple 2

Pour des raisons de débogage, vous voulez vérifier si le deuxième paramètre ($2) d'une méthode est un pointeur vers une table. Le début de votre méthode peut être écrit ainsi :

```
  ` ...
Si(◊Débogage)
   RESOUDRE POINTEUR($2;$vaNomVar;$vlNumTable;$vlNumChamp)
   Si(Non((($vlNumTable>0)&($vlNumChamp=0)&($vlNomVar=""))))
  ` ATTENTION : Le pointeur n'est pas une référence à une table
      TRACE
   Fin de si
Fin de si
  ` ...
```

Exemple 3

Reportez-vous à l'exemple de la commande **PROPRIETES GLISSER DEPOSER**.

Exemple 4

Voici un exemple de pointeur vers un tableau 2D :

```
TABLEAU TEXTE(atCities;100;50)
C_POINTEUR($city)
atCities{1}{2}:="Rome"
atCities{1}{5}:="Paris"
atCities{2}{6}:="New York"
 // ...autres valeurs
$city:=->atCities{1}{5}
RESOUDRE POINTEUR($city;$var;$rowNum;$colNum)
 //$var="atCities"
 //$rowNum="1"
 //$colNum="5"
```

Self

Self -> Résultat

Paramètre	Type		Description
Résultat	Pointeur		Pointeur vers l'objet du formulaire dont la méthode est en cours d'exécution (le cas échéant) Sinon Nil (->[]) si hors contexte

Note de compatibilité

Cette commande est conservée pour des raisons de compatibilité uniquement. A compter de la version 12 de 4D, il est conseillé d'utiliser la commande **OBJET Lire pointeur**.

Description

Self retourne un pointeur vers l'objet du formulaire dont la méthode objet est en cours d'exécution.

La fonction **Self** est utilisée pour référencer une variable dans sa propre méthode objet. Elle retourne un pointeur valide si elle est appelée dans une méthode objet ou dans une méthode projet appelée directement ou indirectement par un méthode objet.

Si **Self** est appelée en-dehors de ce contexte, elle retourne un pointeur **Nil**(->*[]*).

Conseil : Self est très utile lorsque plusieurs objets d'un formulaire doivent effectuer la même action, opérée sur eux-mêmes.

Note : Lorsqu'elle est utilisée avec une list box, la fonction **Self** retourne un pointeur vers la list box ou la colonne de la list box en fonction du contexte. Pour plus d'informations, reportez-vous à la section **Gestion programmée des objets de type List box**.

Exemple

Référez-vous à l'exemple de la commande **RESOUDRE POINTEUR**.

TRACE

TRACE
Ne requiert pas de paramètre

Description

La commande **TRACE** est utilisée, lors du développement des bases, pour tracer des méthodes, c'est-à-dire contrôler leur exécution pas à pas.

La commande **TRACE** affiche le débogueur de 4D dans le process courant. La fenêtre du **Débogueur** apparaît dès que la commande est appelée, avant l'exécution de la ligne de code suivante, et reste affichée pour l'exécution de chaque ligne de code. Vous pouvez également appeler manuellement le débogueur en utilisant la combinaison **Alt+Maj+clic droit** sous Windows ou **Control+Option+Commande+clic** sous Mac OS pendant l'exécution du code.

Dans les bases compilées, cette commande est ignorée.

4D Server : Si vous appelez **TRACE** depuis une méthode projet exécutée en tant que Procédure stockée, la fenêtre du débogueur apparaîtra sur le poste serveur.

Conseil : N'appelez pas **TRACE** lorsque vous utilisez un formulaire pour lequel les événements Sur activation et Sur désactivation ont été sélectionnés. En effet, chaque fois que la fenêtre du débogueur apparaîtra, les événements formulaire seront activés et cela créera une boucle sans fin entre les événements et le débogueur. De même, si vous appelez la commande **TRACE** depuis une méthode formulaire ou objet exécutée pendant la mise à jour du formulaire à l'écran, vous devrez également faire face à un problème de répétition sans fin de la séquence mises à jour du formulaire/apparitions de la fenêtre du débogueur.
Si vous vous retrouvez dans une telle situation, pour en sortir, utilisez la combinaison **Maj+clic** sur le bouton **Reprendre exécution** du débogueur. Tous les appels ultérieurs à **TRACE** dans le process seront ignorés.

Exemple

Dans le code suivant, la variable process *CREER_LANG* doit être égale à “US” ou “FR”. Si ce n'est pas le cas, la méthode projet ***DEBUG*** est appelée :

```
   ` ...
Au cas ou
   :(CREER_LANG="US")
      vsBHCmdNom:=[Commandes]CM US Nom
   :(CREER_LANG="FR")
      vsBHCmdNom:=[Commandes]CM FR Nom
   Sinon
      DEBUG("Valeur de CREER_LANG incorrecte")
Fin de cas
```

La méthode projet ***DEBUG*** est listée ci-dessous :

```
   ` Méthode projet DEBUG
   ` DEBUG (Texte)
   ` DEBUG (Informations supplémentaires de débogage)

C_TEXTE($1)

Si(◊vbDebugOn) ` Variable interprocess définie dans la méthode base Sur ouverture
   Si(Mode compile)
      Si(Nombre de paramètres>=1)
         ALERTE($1+Caractere(13)+"Appelez le concepteur au 05 05 05 05")
      Fin de si
   Sinon
      TRACE
   Fin de si
Fin de si
```

Type

Type (champVar) -> Résultat

Paramètre	Type		Description
champVar	Champ, Variable	➡	Champ ou variable à tester
Résultat	Entier long	⮌	Numéro du type de données

Description

Type retourne une valeur numérique qui indique le type du champ ou de la variable que vous avez passé(e) dans le paramètre *champVar*.
4D fournit les constantes prédéfinies suivantes, disponible dans le thème **Types champs et variables** :

Constante	Type	Valeur
Est un BLOB	Entier long	30
Est un booléen	Entier long	6
Est un champ alpha	Entier long	0
Est un entier	Entier long	8
Est un entier 64 bits	Entier long	25
Est un entier long	Entier long	9
Est un float	Entier long	35
Est un null JSON	Entier long	255
Est un numérique	Entier long	1
Est un objet	Entier long	38
Est un pointeur	Entier long	23
Est un tableau 2D	Entier long	13
Est un tableau blob	Entier long	31
Est un tableau booléen	Entier long	22
Est un tableau chaîne	Entier long	21
Est un tableau date	Entier long	17
Est un tableau entier	Entier long	15
Est un tableau entierlong	Entier long	16
Est un tableau heure	Entier long	32
Est un tableau image	Entier long	19
Est un tableau numérique	Entier long	14
Est un tableau objet	Entier long	39
Est un tableau pointeur	Entier long	20
Est un tableau texte	Entier long	18
Est un texte	Entier long	2
Est une date	Entier long	4
Est une heure	Entier long	11
Est une image	Entier long	3
Est une sous table	Entier long	7
Est une variable chaîne	Entier long	24
Est une variable indéfinie	Entier long	5

Notes :

- **Type** retourne 9 (Est un entier long) lorsqu'elle est appliquée à une variable de type Graphe.
- A compter de la version 11 de 4D, **Type** retourne le véritable type d'un tableau lorsqu'elle est appliquée à une "ligne" d'un tableau 2D, et non plus Est un tableau 2D (cf. exemple 4).
- A compter de la version 11 de 4D, **Type** retourne le type Est un texte ou Est un tableau texte lorsqu'elle est appliquée respectivement à une variable Alpha ou un tableau Alpha (à compter de cette version, il n'y a pas de différence entre une variable Alpha et Texte).

Vous pouvez appliquer la fonction **Type** aux champs, variables interprocess, variables process, variables locales et à des pointeurs dépointés qui référencent ces types d'objets. Vous pouvez appliquer **Type** aux paramètres (*$1,$2..., ${...}*) d'une méthode projet ou au résultat d'une fonction (*$0*).

Exemple 1

Référez-vous à l'exemple de la commande **AJOUTER DONNEES AU CONTENEUR**.

Exemple 2

Référez-vous à l'exemple de la commande **PROPRIETES GLISSER DEPOSER** .

Exemple 3

La méthode projet suivante efface une partie ou la totalité des champs de l'enregistrement courant de la table vers laquelle pointe le pointeur passé en paramètre, et ce, sans supprimer l'enregistrement ou changer d'enregistrement courant :

```
` Méthode projet VIDER ENREGISTREMENT
```

```
  ` VIDER ENREGISTREMENT ( Pointeur {; Entier long } )
  ` VIDER ENREGISTREMENT ( -> [Table] { ; Type des valeurs } )

C_POINTEUR($1)
C_ENTIER LONG($2;$vlTypeVal)

Si(Nombre de paramètres>=2)
   $vlTypeVal:=$2
Sinon
   $vlTypeVal:=0xFFFFFFFF
Fin de si
Boucle($vlChamp;1;Nombre de champs($1))
   $vpChamp:=Champ(Table($1);$vlChamp)
   $vlTypeChamp:=Type($vpChamp->)
   Si($vlTypeVal ??$vlTypeChamp )
      Au cas ou
         :(($vlTypeChamp=Est un champ alpha)|($vlTypeChamp=Est un texte))
            $vpChamp->:=""
         :(($vlTypeChamp=Est un numérique)|($vlTypeChamp=Est un entier)|($vlTypeChamp=Est un entier long))
            $vpChamp->:=0
         :($vlTypeChamp=Est une date)
            $vpChamp->:=!00/00/00!
         :($vlTypeChamp=Est une heure)
            $vpChamp->:=?00:00:00?
         :($vlTypeChamp=Est un booléen)
            $vpChamp->:=Faux
         :($vlTypeChamp=Est une image)
            C_IMAGE($vgImageVide)
            $vpChamp->:=$vgImageVide
         :($vlTypeChamp=Est une sous table)
            Repeter
               TOUS LES SOUS ENREGISTREMENTS($vpChamp->)
               SUPPRIMER SOUS ENREGISTREMENT($vpChamp->)
            Jusque(Sous enregistrements trouves($vpChamp->)=0)
         :($vlTypeChamp=Est un BLOB)
            FIXER TAILLE BLOB($vpChamp->;0)
      Fin de cas
   Fin de si
Fin de boucle
```

Une fois cette méthode projet implémentée dans votre base, vous pouvez écrire :

```
  ` Effacer la totalité du contenu de l'enregistrement courant de la table [Choses à faire]
VIDER ENREGISTREMENT(->[Choses à faire])

  ` Effacer les champs de type Texte, BLOB et Image de l'enregistrement courant de la table [Choses à faire]
VIDER ENREGISTREMENT(->[Choses à faire];0?+Est un texte?+Est un BLOB?+Est une image)

  ` Effacer la totalité de l'enregistrement courant de la table [Choses à faire] sauf les champs Alpha
VIDER ENREGISTREMENT(->[Choses à faire];-1?-Est un champ alpha)
```

Exemple 4

Dans certains cas, par exemple pour écrire du code générique, il peut être nécessaire de savoir si un tableau est tableau standard indépendant ou une “ligne” d’un tableau 2D. Dans ce cas, il suffit d’utiliser le code suivant :

```
ptrmonTab:=->monTab{6} ` monTab{6} est-il une ligne d’un tableau 2D ?
RESOUDRE POINTEUR(ptrmonTab;nomVar;numTable;numChamp)
Si(nomVar#"")
   $ptr:=Pointeur vers(nomVar)
   $letype:=Type($ptr->)
  ` Si monTab{6} est une ligne de tableau 2D, $letype vaut 13
Fin de si
```

LDAP

Présentation des commandes LDAP

Les commandes du thème "LDAP" permettent à votre application 4D de se connecter à un annuaire d'entreprise tel que MS Active Directory à l'aide de LDAP. Vous pouvez accéder aux données du serveur y effectuer des recherches.

Note : LDAP ou Lightweight Directory Access Protocol est un standard pour l'accès et la maintenance de services d'information distribuée. Pour plus d'informations, veuillez vous reporter à la page Wikipedia consacrée à LDAP ou la page principale OpenLDAP Software.

Les commandes LDAP vous permettent notamment :

- d'utiliser le login et le mot de passe de la session Windows pour l'accès à votre application 4D (si vous utilisez MS Active Directory) de manière à ce que l'utilisateur n'ait qu'à mémoriser un seul mot de passe,
- d'interroger l'annuaire d'entreprise afin de récupérer des informations utilisateur telles que le nom complet, l'email, le numéro de téléphone, les groupes auquel il appartient, etc.

Dans 4D, une connexion LDAP est ouverte à l'aide de la commande **LDAP LOGIN**. Elle est ensuite associée au process 4D courant et sera refermée à l'aide de **LDAP LOGOUT**, ou lorsque le process terminera son exécution.

Glossaire

Voici la liste des principaux acronymes utilisés dans l'environnement LDAP :

Acronyme	Définition
LDAP	Lightweight Directory Access Protocol
AD	Active Directory. AD est une base d'annuaire d'entreprise implémentée par Microsoft, et LDAP est un de ses protocoles d'échange.
CN	Common Name, par exemple "John Doe"
DN	Distinguished Name, par exemple "cn=John Doe,ou=users,dc=example,dc=com"
SAM-Account-Name	Security Account Manager, nom utilisé pour la connexion AD, par exemple "jdoe"
OU	Organizational unit, groupes de l'arbre serveur
DC	Domain components, racine et premières branches de l'arbre serveur
uid	User identifier

LDAP Chercher

LDAP Chercher (dnRootEntry ; filtre {; scope {; attributs {; attributsEnTableau}}}) -> Résultat

Paramètre	Type		Description
dnRootEntry	Chaîne	→	Distinguished Name de l'élément racine où démarrer la recherche
filtre	Chaîne	→	Filtre de recherche LDAP
scope	Chaîne	→	Champ d'action de la recherche : "base" (défaut), "one" ou "sub"
attributs	Tableau texte	→	Attribut(s) à récupérer
attributsEnTableau	Tableau booléen	→	Vrai = forcer le retour des attributs en tableaux, Faux = forcer le retour des attributs en variables simples
Résultat	Objet	←	Attributs clé/valeur

Description

La commande **LDAP Chercher** recherche sur le serveur LDAP cible la première occurrence correspondant aux critères définis. Cette commande doit être exécutée dans le contexte d'une connexion serveur LDAP ouverte par la commande **LDAP LOGIN** dans le process courant ; sinon une erreur 1003 est retournée.

Dans *dnRootEntry*, passez le *Distinguished Name* de l'élément racine du serveur LDAP ; la recherche démarrera à partir de cet élément.

Dans *filtre*, passez le filtre de recherche LDAP à appliquer. Ce filtre doit être conforme à la rfc2225. Vous pouvez passer une chaîne vide "" afin de ne pas appliquer de filtre. Le joker "*" pour chercher des sous-chaînes est pris en charge.

Dans *scope*, passez une des constantes suivantes du thème "**LDAP**" :

Constante	Type	Valeur	Comment
LDAP racine et suivant	Chaîne	one	Chercher dans l'élément racine défini par *dnRootEntry* et dans les branches directement suivantes sur un niveau
LDAP racine uniquement	Chaîne	base	Chercher uniquement dans l'élément racine défini par *dnRootEntry* (défaut si omis)
LDAP tous niveaux	Chaîne	sub	Chercher dans l'élément racine défini par *dnRootEntry* et dans toutes les branches suivantes

Dans *attributs*, passez un tableau texte contenant la liste de tous les attributs LDAP à récupérer à partir des entrées trouvées. Par défaut, si ce paramètre est omis, tous les attrbiuts sont récupérés.

Note : Les noms d'attributs LDAP tiennent compte des majuscules/minuscules. Pour plus d'informations sur les attributs LDAP, vous pouvez consulter cette page qui liste tous les attributs disponibles pour MS Active Directory.

Par défaut, la commande retourne les attributs sous forme de tableau si plusieurs résultats sont trouvés, ou sous forme de variable simple si un seul résultat est trouvé. Le paramètre optionnel *attributsEnTableau* vous permet de "forcer" le formatage des attributs retournés en tableau ou en variable pour chaque attribut défini :

- Lorsque vous passez **true** dans un élément, l'élément correspondant du paramètre *attributs* sera retourné en tableau. Si une seule valeur est trouvée, la commande retourne un tableau à un seul élément.
- Lorsque vous passez **false** dans un élément, l'élément correspondant du paramètre *attributs* sera retourné en variable simple. Si plusieurs valeurs sont trouvées, la commande retourne uniquement le premier élément.

Exemple 1

Vous souhaitez obtenir le numéro de téléphone de l'utilisateur "smith" dans l'annuaire de l'enterprise :

```
TABLEAU TEXTE($_tabAttributes;0)
AJOUTER A TABLEAU($_tabAttributes;"cn")
AJOUTER A TABLEAU($_tabAttributes;"phoneNumber")
LDAP LOGIN($url;$dn;$pwd)
$filter:="cn=*smith*"
$vfound:=LDAP Chercher($dnSearchRootEntry;$filter;LDAP tous niveaux;$_tabAttributes)
LDAP LOGOUT
```

Exemple 2

Vous voulez obtenir un tableau de toutes les entrées trouvées pour l'attribut "memberOf" :

```
C_OBJET($entry)
TABLEAU TEXTE($_tabAttributes;0)
TABLEAU BOOLEEN($_tabAttributes_asArray;0)
AJOUTER A TABLEAU($_tabAttributes;"cn")
AJOUTER A TABLEAU($_tabAttributes_asArray;Faux)
AJOUTER A TABLEAU($_tabAttributes;"memberOf")
AJOUTER A TABLEAU($_tabAttributes_asArray;Vrai)

LDAP LOGIN($url;$login;$pwd;LDAP mot de passe en clair)
$entry:=LDAP Chercher($dnSearchRootEntry;"cn=adrien*";LDAP tous niveaux;$_tabAttributes;$_tabAttributes_asArray)
LDAP LOGOUT

TABLEAU TEXTE($_arrMemberOf;0)
OB LIRE TABLEAU($entry;"memberOf";$_arrMemberOf)
 // $_arrMemberOf est un tableau contenant tous les groupes de l'entrée
```

LDAP CHERCHER TOUS

LDAP CHERCHER TOUS (dnRootEntry ; tabRésultat ; filtre {; scope {; attributs {; attributsEnTableau}}})

Paramètre	Type		Description
dnRootEntry	Chaîne	➡	Distinguished Name de l'élément racine où démarrer la recherche
tabRésultat	Tableau objet	⬅	Résultat de la recherche
filtre	Chaîne	➡	Filtre de recherche LDAP
scope	Chaîne	➡	Champ d'action de la recherche : "base" (défaut), "one" ou "sub"
attributs	Tableau texte	➡	Attribut(s) à récupérer
attributsEnTableau	Tableau booléen	➡	Vrai = forcer le retour des attributs en tableaux, Faux = forcer le retour des attributs en variables simples

Description

La commande **LDAP CHERCHER TOUS** recherche sur le server LDAP cible toutes les occurrences correspondant aux critères définis. Cette commande doit être exécutée dans le contexte d'une connexion serveur LDAP ouverte par la commande **LDAP LOGIN** dans le process courant ; sinon une erreur 1003 est retournée.

A noter que les serveurs LDAP imposent généralement un nombre maximum d'entrées qui peuvent être récupérées lors d'une recherche. Par exemple, Microsoft Active directory limite de nombre à 1000 entrées par défaut.

Dans *dnRootEntry*, passez le *Distinguished Name* de l'élément racine du serveur LDAP ; la recherche démarrera à partir de cet élément.

Dans *tabResult*, passez un tableau objet qui sera rempli avec les entrées trouvées ; dans ce tableau, chaque élément est un objet contenant les paires attributs/valeurs retournées pour une entrée trouvée. Vous pouvez utiliser le paramètre *attributs* pour définir les paramètres à retourner.

Dans *filtre*, passez le filtre de recherche LDAP à appliquer. Ce filtre doit être conforme à la rfc2225. Vous pouvez passer une chaîne vide "" afin de ne pas appliquer de filtre. Le joker "*" pour chercher des sous-chaînes est pris en charge.

Dans *scope*, passez une des constantes suivantes du thème "**LDAP**" :

Constante	Type	Valeur	Comment
LDAP racine et suivant	Chaîne	one	Chercher dans l'élément racine défini par *dnRootEntry* et dans les branches directement suivantes sur un niveau
LDAP racine uniquement	Chaîne	base	Chercher uniquement dans l'élément racine défini par *dnRootEntry* (défaut si omis)
LDAP tous niveaux	Chaîne	sub	Chercher dans l'élément racine défini par *dnRootEntry* et dans toutes les branches suivantes

Dans *attributs*, passez un tableau texte contenant la liste de tous les attributs LDAP à récupérer à partir des entrées trouvées. Par défaut, si ce paramètre est omis, tous les attributs sont récupérés.

Note : Les noms d'attributs LDAP tiennent compte des majuscules/minuscules. Pour plus d'informations sur les attributs LDAP, vous pouvez consulter cette page qui liste tous les attributs disponibles pour MS Active Directory.

Par défaut, la commande retourne les attributs sous forme de tableau si plusieurs résultats sont trouvés, ou sous forme de variable simple si un seul résultat est trouvé. Le paramètre optionnel *attributsEnTableau* vous permet de "forcer" le formatage des attributs retournés en tableau ou en variable pour chaque attribut défini :

- Lorsque vous passez **true** dans un élément, l'élément correspondant du paramètre *attributs* sera retourné en tableau. Si une seule valeur est trouvée, la commande retourne un tableau à un seul élément.
- Lorsque vous passez **false** dans un élément, l'élément correspondant du paramètre *attributs* sera retourné en variable simple. Si plusieurs valeurs sont trouvées, la commande retourne uniquement le premier élément.

Exemple 1

Nous voulons récupérer les numéros de téléphone de tous les utilisateurs nommés "smith" dans l'annuaire d'enterprise :

```
TABLEAU TEXTE($_tabAttributes;0)
TABLEAU BOOLEEN($_tabAttributes_asArray;0)
AJOUTER A TABLEAU($_tabAttributes;"cn")
AJOUTER A TABLEAU($_tabAttributes_asArray;Faux)
AJOUTER A TABLEAU($_tabAttributes;"telephoneNumber")
AJOUTER A TABLEAU($_tabAttributes_asArray;Faux)
TABLEAU OBJET($_entry;0)

LDAP LOGIN($url;$myLogin;$pwd)
$filter:="cn=*smith*"
LDAP CHERCHER TOUS($dnSearchRootEntry;$_entry;$filter;LDAP tous niveaux;$_tabAttributes)
LDAP LOGOUT


 //$_entry contiendra par exemple
 // $_entry{1} = {"cn":"John Smith","telephoneNumber":"01 40 87 00 00"}
 // $_entry{2} = {"cn":"Adele Smith","telephoneNumber":"01 40 87 00 01"}
 // $_entry{3} = {"cn":"Adrian Smith","telephoneNumber":"01 23 45 67 89"}
 // ...
```

Exemple 2

Ces exemples illustrent plus particulièrement l'utilisation du paramètre *attributsEnTableau* :

```
TABLEAU OBJET($_entry;0)
TABLEAU TEXTE($_tabAttributes;0)
TABLEAU BOOLEEN($_tabAttributes_asArray;0)
AJOUTER A TABLEAU($_tabAttributes;"cn")
AJOUTER A TABLEAU($_tabAttributes_asArray;Faux)
AJOUTER A TABLEAU($_tabAttributes;"memberOf")
AJOUTER A TABLEAU($_tabAttributes_asArray;Vrai)

LDAP LOGIN($url;$login;$pwd;LDAP password plain text)
LDAP CHERCHER TOUS($dnSearchRootEntry;$_entry;$filter;LDAP tous niveaux;$_tabAttributes;$_tabAttributes_asArray)
LDAP LOGOUT

TABLEAU TEXTE($_arrMemberOf;0)
OB LIRE TABLEAU($_entry{1};"memberOf";$_arrMemberOf)
 // $_arrMemberOf est un tableau contenant tous les groupes de l'entrée
```

```
TABLEAU TEXTE($_tabAttributes;0)
TABLEAU BOOLEEN($_tabAttributes_asArray;0)
AJOUTER A TABLEAU($_tabAttributes;"cn")
AJOUTER A TABLEAU($_tabAttributes_asArray;Faux)
AJOUTER A TABLEAU($_tabAttributes;"memberOf")
AJOUTER A TABLEAU($_tabAttributes_asArray;Faux)

LDAP LOGIN($url;$login;$pwd;LDAP password plain text)
LDAP CHERCHER TOUS($dnSearchRootEntry;$_entry;$filter;LDAP tous niveaux;$_tabAttributes;$_tabAttributes_asArray)
LDAP LOGOUT

$memberOf:=OB Lire($_entry{1};"memberOf")
 // $memberOf est une variable contenant le premier groupe de l'entrée
```

LDAP LOGIN

LDAP LOGIN (url ; login ; motDePasse {; digest})

Paramètre	Type		Description
url	Chaîne	➡	URL du serveur LDAP auquel se connecter
login	Chaîne	➡	Compte de l'utilisateur
motDePasse	Chaîne	➡	Mot de passe de l'utilisateur
digest	Entier long	➡	0 = envoyer mot de passe en digest MD5 (défaut), 1 = envoyer mot de passe sans encryptage

Description

La commande **LDAP LOGIN** ouvre une connexion en lecture seule sur le serveur LDAP désigné par le paramètre *url* avec les identifiants *login* et *motDePasse* fournis. Si elle est acceptée par le serveur, cette connexion sera utilisée pour toutes les recherches LDAP effectuées par la suite dans le process courant, jusqu'à ce que la commande **LDAP LOGOUT** soit exécutée (ou que le process soit terminé).

Dans *url*, passez l'URL complet du serveur LDAP auquel se connecter, incluant le *scheme* et le port (389 by default). Ce paramètre doit être conforme à la rfc2255.
Vous pouvez ouvrir une connexion sécurisée via TLS en passant un *url* qui débute par "ldaps" et qui utilise un numéro de port spécifique (par exemple "ldaps://svr.ldap.acme.com:1389"). Le serveur LDAP doit généralement disposer d'un certificat SSL (c'est le cas pour MS Active Directory). Il est fortement recommandé d'utiliser une connexion TLS lorsque le mot de passe est transmis en texte brut (voir ci-dessous).

Note : Si vous passez une chaîne vide dans le paramètre *url*, la commande tentera de se connecter au serveur LDAP par défaut disponible sur le domaine (cette fonction est destinée uniquement aux besoins liés aux tests, pour des raisons de performances elle ne doit pas être utilisée en production).

Dans *login*, passez le compte utilisateur sur le serveur LDAP et dans *motDePasse*, passez le mot de passe du compte. Le *login* peut prendre l'une des formes suivantes, en fonction de la configuration du serveur LDAP :

- un Distinguished Name (DN), par exemple "CN=John Smith,OU=users,DC=example,DC=com"
- un nom d'utilisateur (CN), par exemple "CN=John Smith"
- une adresse email, par exemple "johnsmith@4d.fr"
- un SAM-Account-Name, par exemple "jsmith".

Notez que les valeurs admises pour le *login* sont liées au mode de transmission du mot de passe, défini par le paramètre *digest*. Par exemple, dans une configuration par défaut de MS Active Directory :

- Lorsque le mode de transmission est LDAP mot de passe en digest MD5, la seule valeur acceptée pour une connexion utilisateur est le SAM-Account-Name.
- Lorsque le mode de transmission est LDAP mot de passe en clair (texte brut), le paramètre *login* peut contenir soit un DN, un CN ou une adresse email. Un SAM-Account-Name peut être utilisé s'il est précédé du nom de domaine (par exemple "dc-acme.com/jsmith").

Le paramètre *digest* vous permet de modifier le mode de transmission du mot de passe sur le réseau. Vous pouvez utiliser une des constantes suivantes, placées dans le thème "**LDAP**" :

Constante	Type	Valeur	Comment
LDAP mot de passe en clair	Entier long	1	Envoi du mot de passe sans encryptage (connexion TLS recommandée)
LDAP mot de passe en digest MD5	Entier long	0	(Défaut) Envoi du mot de passe encrypté en MD5

Par défaut, le *motDePasse* est transmis en digest MD5. Passez LDAP mot de passe en clair si nécessaire, par exemple si vous voulez utiliser des chaînes de *login* différentes avec le serveur LDAP. Dans un environnement de production, il est fortement recommandé dans ce cas de recourir à une connexion TLS dans le paramètre *url*.

Note : L'authentification avec un mot de passe vide vous permet d'activer le mode de connexion anonyme (s'il est autorisé par le serveur LDAP). Cependant, des erreurs pourront être générées si vous essayez d'effectuer des opérations non autorisées dans ce mode spécifique.

Si les paramètres de connexion sont valides, une connexion au serveur LDAP est ouverte dans le process 4D. Vous pouvez alors rechercher et récupérer des informations à l'aide des commandes LDAP.

N'oubliez pas d'appeler la commande **LDAP LOGOUT** lorsque la connexion au serveur LDAP n'est plus nécessaire.

Exemple 1

Vous voulez vous connecter à un serveur LDAP et effectuer une recherche :

```
TABLEAU TEXTE($_tabAttributes;0)
AJOUTER A TABLEAU($_tabAttributes;"cn")
AJOUTER A TABLEAU($_tabAttributes;"phoneNumber")
LDAP LOGIN("ldap://srv.dc.acme.com:389";"John Smith";"qrnSurBret2elburg")
$vfound:=LDAP Chercher("OU=UO_Users,DC=ACME,DC=com";cn=John Doe";LDAP tous niveaux;$_tabAttributes)
LDAP LOGOUT //ne pas oublier de se déconnecter
```

Exemple 2

Cet exemple tente de se connecter à une application :

```
APPELER SUR ERREUR("ErrHdlr") //gestion d'erreurs
errOccured:=Faux
errMsg:=""
Si(ppBindMode=1) //si mot de passe transmis en mode par défaut
   LDAP LOGIN(vUrlLdap;vUserCN;vPwd;LDAP mot de passe en digest MD5)
Sinon
```

```
   LDAP LOGIN(vUrlLdap;vUserCN;vPwd;LDAP mot de passe en clair)
Fin de si

Au cas ou
   :(Non(errOccurred))
      ALERTE("Vous êtes connecté au serveur LDAP. ")

   :(errOccurred)
      ALERTE("Erreurs dans les paramètres")
Fin de cas

LDAP LOGOUT
APPELER SUR ERREUR("")
```

LDAP LOGOUT

LDAP LOGOUT
Ne requiert pas de paramètre

Description

La commande **LDAP LOGOUT** referme la connexion LDAP active dans le process courant (le cas échéant). S'il n'y a pas de connexion avec un server LDAP, l'erreur 1003 indiquant que vous n'êtes pas connecté est retournée.

Liens

Présentation des liens

Les commandes de ce thème, en particulier **CHARGER SUR LIEN** et **LIEN RETOUR**, établissent et gèrent les relations entre les tables, à la fois pour les liens automatiques et les liens manuels. Consultez le manuel Mode Développement de 4D pour la création des liens entre les tables avant d'utiliser ces commandes.

Exploiter par programmation les liens automatiques entre les tables

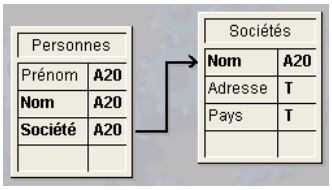
Deux tables peuvent être reliées par un lien automatique. En général, quand un lien automatique est créé, les enregistrements liés sont chargés et sélectionnés dans la table liée. Un grand nombre d'opérations exploitent cette relation. En particulier, citons les opérations suivantes :

- Saisie de données,
- Liste des enregistrements à l'écran dans un formulaire de sortie,
- Etats,
- Opérations sur une sélection d'enregistrements comme les recherches, les tris et les formules.

Pour améliorer les performances, quand 4D active les liens automatiques, seul un enregistrement devient l'enregistrement courant pour la table. Pour chacune des opérations énumérées ci-dessus, l'enregistrement lié est chargé selon les principes suivants :

- Si un lien sélectionne un seul enregistrement de la table liée, cet enregistrement est chargé du disque.
- Si un lien sélectionne plusieurs enregistrements de la table liée, une nouvelle sélection d'enregistrements est créée pour cette table, et le premier enregistrement de cette sélection est chargé du disque.

Par exemple, dans la structure affichée ci-dessous, si un enregistrement pour la table [Personnes] est chargé et affiché pour une saisie de donnés, l'enregistrement lié dans la [Sociétés] est sélectionné et chargé. De la même façon, si un enregistrement pour la table [Sociétés] est chargé et affiché pour de la saisie de données, les enregistrements liés de la table [Personnes] sont sélectionnés.



Dans le type de schéma présenté ci-dessus, la table [Personnes] est appelée **Table N** et la table [Sociétés] est appelée **Table 1**. Pour vous souvenir de la différence, pensez à “plusieurs personnes travaillent dans une société” ou “une société emploie N personnes”.

De même, le champ Société de la table [Personnes] est appelé **Champ N**, et le champ Nom de la table [Sociétés] est appelé **Champ 1**.

Il n'est pas toujours possible que le champ lié soit unique. Par exemple, le champ [Sociétés]Nom peut contenir des noms de sociétés identiques. Ce cas de non-unicité est facilement contournable : il suffit de créer un lien vers un autre champ de la table liée qui, lui, sera toujours unique. Ce champ pourrait être par exemple un numéro d'identification de la société.

Les commandes listées ci-dessous utilisent les liens automatiques pour charger les enregistrements liés lors de leur exécution. Toutes ces commandes activent les liens automatiques "de N vers 1" existants. En revanche, seules les commandes signalées explicitement par un Oui activent les liens automatiques "de 1 vers N".

Commande	Lien 1 vers N
AJOUTER ENREGISTREMENT	Oui
_o_AJOUTER SOUS ENREGISTREMENT	Non
APPLIQUER A SELECTION	Non
VISUALISER SELECTION	Non
ECRITURE DIF	Non
ECRITURE SYLK	Non
EXPORTER TEXTE	Non
EXPORTER DONNEES	Non
MODIFIER ENREGISTREMENT	Oui
_o_MODIFIER SOUS ENREGISTREMENT	Non
MODIFIER SELECTION	Oui (en saisie de données)
TRIER	Non
TRIER PAR FORMULE	Non
CHERCHER PAR FORMULE	Oui
CHERCHER DANS SELECTION	Oui
CHERCHER	Oui
IMPRIMER ETIQUETTES	Non
IMPRIMER SELECTION	Oui
QR ETAT	Non
SELECTION VERS TABLEAU	Non
SELECTION LIMITEE VERS TABLEAU	Non

Activer par programmation les liens entre les tables

Que les liens soient automatiques ne signifie pas que les enregistrements liés ou les enregistrements pour une table seront sélectionnés simplement parce qu'une commande charge un enregistrement. Après avoir exécuté une commande chargeant un enregistrement, dans certains cas vous devez explicitement

appeler le ou les enregistrement(s) lié(s) avec **CHARGER SUR LIEN** ou **LIEN RETOUR** si vous avez besoin d'accéder aux données liées.

Certaines des commandes listées ci-dessus (par exemple les commandes de recherche) chargent l'enregistrement courant une fois leur tâche terminée. Dans ce cas, l'enregistrement chargé n'appelle pas automatiquement le ou les enregistrement(s) lié(s). Là aussi, vous devez explicitement sélectionner le ou les enregistrement(s) lié(s) avec **CHARGER SUR LIEN** ou **LIEN RETOUR** si vous avez besoin d'accéder aux données liées.

ANCIEN LIEN RETOUR

ANCIEN LIEN RETOUR (leChamp)

Paramètre	Type		Description
leChamp	Champ	⇨	Champ recevant un lien

Description

ANCIEN LIEN RETOUR fonctionne comme la commande **LIEN RETOUR**, à la différence près que **ANCIEN LIEN RETOUR** utilise l'ancienne valeur du champ pour activer le lien.

Note : ANCIEN LIEN RETOUR utilise l'ancienne valeur du champ N, telle qu'elle est retournée par la fonction **Ancien**. Reportez-vous à la description de cette fonction pour plus d'informations.

ANCIEN LIEN RETOUR modifie la sélection de la table liée. La commande sélectionne le premier enregistrement de la sélection courante et en fait l'enregistrement courant.

CHARGER ANCIEN

CHARGER ANCIEN (leChamp)

Paramètre	Type		Description
leChamp	Champ	⇨	Champ N

Description

CHARGER ANCIEN fonctionne de la même manière que **CHARGER SUR LIEN**, à la différence près que **CHARGER ANCIEN** utilise la valeur précédente de *leChamp* pour établir la relation.

Note : CHARGER ANCIEN utilise l'ancienne valeur du champ N telle qu'elle est retournée par la fonction **Ancien**. Reportez-vous à la description de cette fonction pour plus d'informations.

CHARGER ANCIEN charge l'enregistrement précédemment lié à l'enregistrement courant. Les champs de cet enregistrement sont alors saisissables. Si vous voulez modifier cet ancien enregistrement lié et le sauvegarder, vous devez appeler la commande **STOCKER SUR LIEN**. Notez que pour un enregistrement venant d'être créé, il n'y pas d'ancien enregistrement lié.

Variables et ensembles système

Si la commande a été correctement exécutée et si les enregistrements liés ont bien été chargés, la variable système OK prend la valeur 1. Si l'utilisateur a cliqué sur le bouton **Annuler** dans la boîte de dialogue de choix d'enregistrement (qui apparaît si l'enregistrement lié avait été modifié), la variable OK prend la valeur 0.

CHARGER SUR LIEN

CHARGER SUR LIEN (tableN | champN {; discriminant})

Paramètre	Type		Description
tableN | champN	Table, Champ	➡	Table pour laquelle définir tous les liens automatiques ou Champ avec lien manuel partant vers la table 1
discriminant	Champ	➡	Champ discriminant de la table 1

Description

CHARGER SUR LIEN accepte deux syntaxes.

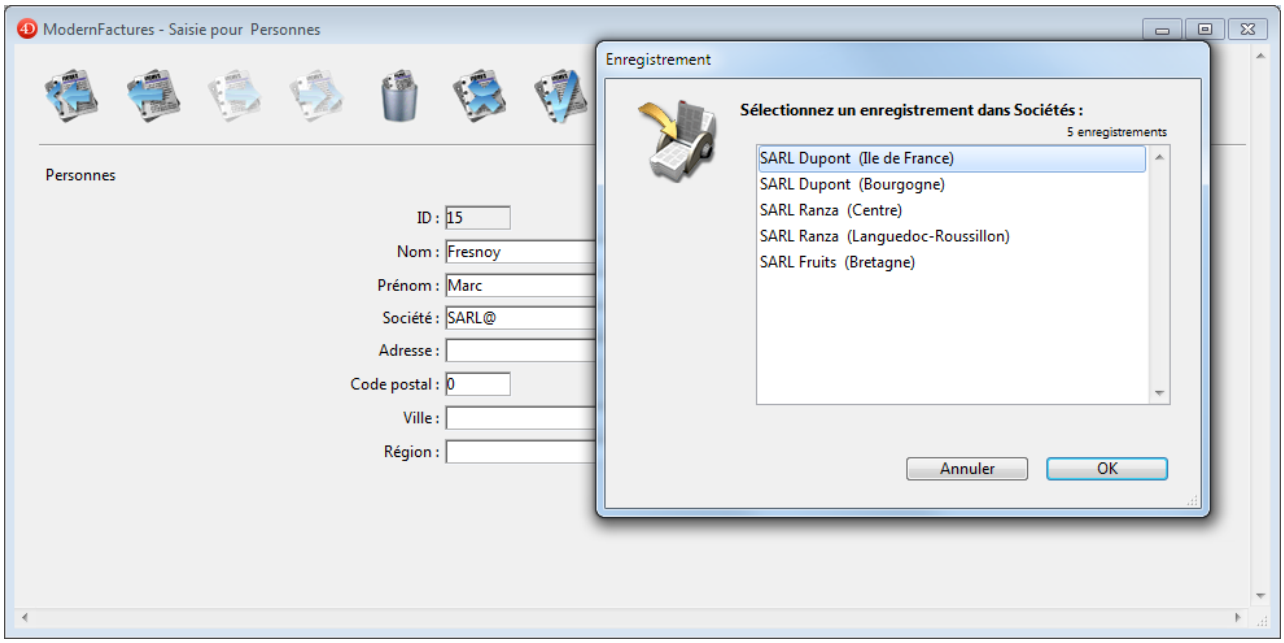
La première syntaxe de la commande, **CHARGER SUR LIEN(tableN)**, active tous les liens aller automatiques (de N vers 1) pour la table *tableN* dans le process courant. Cela signifie que pour chaque champ de la *tableN* d'où part un lien aller automatique, la commande sélectionnera l'enregistrement lié dans chaque table liée. Cela peut donc modifier l'enregistrement courant dans la (les) table(s) liée(s) du process courant.

La seconde syntaxe, **CHARGER SUR LIEN(champN{;discriminant})**, recherche l'enregistrement lié au champ *champN*. Il n'est pas nécessaire que le lien soit automatique. S'il existe, **CHARGER SUR LIEN** charge en mémoire l'enregistrement lié, et en fait l'enregistrement et la sélection courants de la table à laquelle il appartient.

Le paramètre optionnel *discriminant* doit être un champ de la table liée. Il peut être uniquement de type Alpha, Texte, numérique, Date, Heure ou Booléen. En particulier, il ne peut pas être de type Image ou Blob. Si *champN* est spécifié, et si plus d'un enregistrement est trouvé dans la table liée, **CHARGER SUR LIEN** affiche une liste des enregistrements qui correspondent à la valeur de *champN*, permettant à l'utilisateur de sélectionner un enregistrement. Dans cette liste, la colonne de gauche affiche les valeurs des champs liés, la colonne de droite affiche les valeurs de *discriminant*.

Généralement, plusieurs enregistrements sont trouvés lorsque *champN* se termine par le caractère Joker (@). S'il n'y en a qu'un seul, la liste de sélection n'apparaît pas.

Dans l'écran ci-dessous, un enregistrement est en train d'être saisi et une liste de sélection s'affiche au premier plan.



La commande suivante a fait apparaître la liste de sélection :

```
CHARGER SUR LIEN([Personnes]Société;[Sociétés]Région)
```

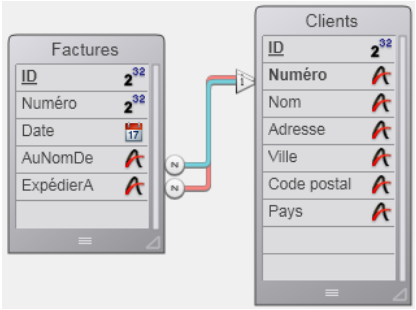
L'utilisateur a saisi SARL@ pour visualiser la liste de toutes les sociétés dont le nom commence par SARL, ainsi que leur région.

Spécifier un champ dans *discriminant* est la même opération que celle qui consiste à définir un champ discriminant dans la boîte de dialogue de définition des propriétés d'un lien en mode Développement. Pour plus d'informations sur la définition d'un champ discriminant, reportez-vous au manuel *Mode Développement* de 4D.

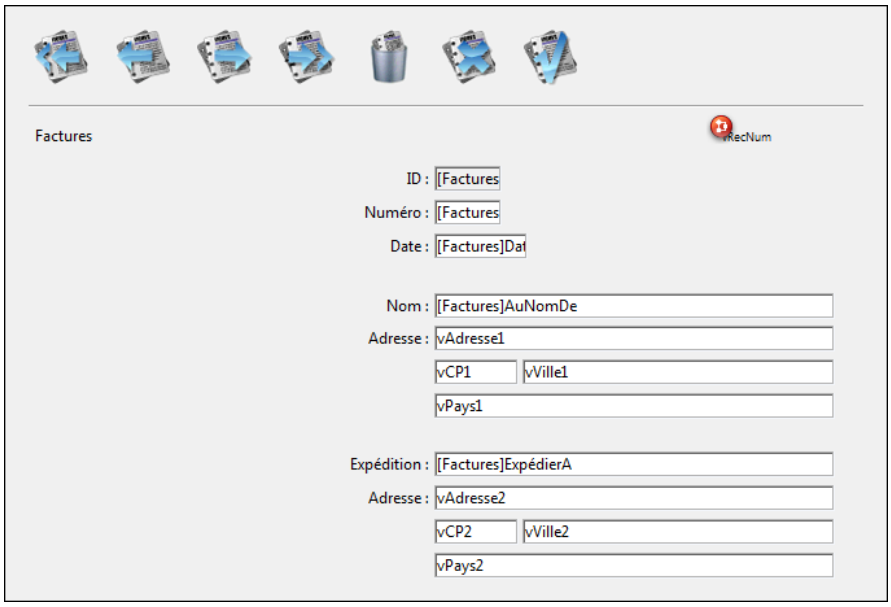
Note : Cette commande ne prend pas en charge les champs de type Objet.

Exemple

Dans l'exemple suivant, la table [Factures] est reliée à la table [Clients] par deux liens manuels. Un lien part du champ [Factures]AuNomDe et va vers le champ [Clients]Numéro, l'autre lien va de [Factures]ExpédierA à [Clients]Numéro.



Voici le formulaire de la table [Factures] affichant les informations "AuNomDe" et "ExpédierA".



Comme les deux liens pointent vers la même table, [Clients], l'information qu'ils récupèrent doit être affichée dans des variables. Si le formulaire contenait les champs de [Clients], seules les valeurs issues du second lien seraient affichées.

Les deux méthodes suivantes sont les méthodes objet des champs [Factures]ExpédierA et [Factures]AuNomDe. Voici la méthode objet du champ [Factures]AuNomDe :

```
CHARGER SUR LIEN([Factures]AuNomDe;[Clients]Adresse)
vAdresse1:=[Clients]Adresse
vVille1:=[Clients]Ville
vPays1:=[Clients]Pays
vCP1:=[Clients]Code postal
```

Voici la méthode objet du champ [Factures]ExpédierA :

```
CHARGER SUR LIEN([Factures]ExpédierA;[Clients]Adresse)
vAdresse2:=[Clients]Adresse
vVille2:=[Clients]Ville
vPays2:=[Clients]Pays
vCP2:=[Clients]Code postal
```

Variables et ensembles système

Si la commande a été correctement exécutée et si les enregistrements liés ont bien été chargés, la variable système OK prend la valeur 1. Si l'utilisateur a cliqué sur le bouton **Annuler** dans la boîte de dialogue de choix d'enregistrement (qui apparaît si l'enregistrement lié avait été modifié), la variable OK prend la valeur 0.

CREER SUR LIEN

CREER SUR LIEN (leChamp)

Paramètre	Type		Description
leChamp	Champ	⇨	Champ N (champ d'où part le lien)

Description

CREER SUR LIEN a deux effets. S'il n'existe pas d'enregistrement lié à *leChamp* (c'est-à-dire si la valeur courante de *leChamp* n'est présente dans le champ correspondant d'aucun enregistrement de la table liée), **CREER SUR LIEN** crée un nouvel enregistrement lié. Si vous souhaitez conserver dans cet enregistrement la valeur de *leChamp* ayant provoqué sa création, assignez-la au champ correspondant. Utilisez ensuite la commande **STOCKER SUR LIEN** pour sauvegarder le nouvel enregistrement.

Si un enregistrement lié existe déjà, la commande **CREER SUR LIEN** a alors exactement le même effet que **CHARGER SUR LIEN** : elle charge en mémoire l'enregistrement lié.

FIXER LIEN CHAMP

FIXER LIEN CHAMP (tableN | champN ; aller ; retour)

Paramètre	Type		Description
tableN | champN	Table, Champ	⇨	Table de départ des liens ou Champ de départ du lien
aller	Entier long	⇨	Statut du lien aller partant du champ ou des liens aller partant de la table
retour	Entier long	⇨	Statut du lien retour partant du champ ou des liens retour partant de la table

Description

La commande **FIXER LIEN CHAMP** permet de définir séparément le statut automatique/manuel de chaque lien de la base pour le process courant, quel que soit son statut initial défini en mode Développement dans la fenêtre de paramétrage des liens.

Passez dans le premier paramètre un nom de table ou de champ :

- si vous passez un nom de champ (*champN*), la commande s'appliquera uniquement au lien partant du champ N désigné.
- si vous passez un nom de table (*tableN*), la commande s'appliquera à tous les liens partant de la table N désignée.
- si aucun lien ne part du champ *champN* ou de la table *tableN*, l'erreur de syntaxe n°16 ("Ce champ ne possède pas de lien") est générée et la variable système OK prend la valeur 0.

Passez dans les paramètres *aller* et *retour* des valeurs indiquant la modification du statut automatique/manuel à appliquer respectivement au(x) lien(s) de type N vers 1 — c'est-à-dire au(x) lien(s) aller — et au(x) lien(s) de type 1 vers N — c'est-à-dire au(x) lien(s) retour — désigné(s). Vous pouvez utiliser les constantes du thème "**Liens**" :

- Ne pas changer (0) = ne pas modifier le statut courant du ou des lien(s).
- Configuration structure (1) = utiliser le paramétrage défini pour le(s) lien(s) dans la fenêtre de Structure de l'application.
- Manuel (2) = rendre manuel(s) le(s) lien(s) pour le process courant.
- Automatique (3) = rendre automatique(s) le(s) lien(s) pour le process courant.

Note : Les modifications effectuées à l'aide de cette commande s'appliquent au process courant uniquement. Le paramétrage des liens défini à l'aide des options de la fenêtre Inspecteur n'est pas modifié.

Note : Si vous avez passé la valeur Vrai à la commande **FIXER LIENS AUTOMATIQUES** durant la même session, les appels à la commande **FIXER LIEN CHAMP** sont ignorés, qu'ils soient placés avant ou après **FIXER LIENS AUTOMATIQUES**. Pour "déverrouiller" le mode automatique et prendre en compte les appels à **FIXER LIEN CHAMP**, passez Faux à **FIXER LIENS AUTOMATIQUES**.

Exemple

Cette commande simplifie la gestion des liens avec l'éditeur d'états rapides. Dans les versions précédentes de 4D, pour utiliser les liens automatiques autres que ceux définis en mode Développement, il était nécessaire de passer tous les liens en automatique. Désormais, le code suivant permet de n'utiliser que les liens définis :

```
FIXER LIENS AUTOMATIQUES(Faux;Faux) `Initialisation des liens
 `Seuls les liens suivants seront utilisés
FIXER LIEN CHAMP([Facture]ID_Client;Automatique;Automatique)
FIXER LIEN CHAMP([Ligne_Facture]ID_Facture;Automatique;Automatique)
QR ETAT([Facture];Caractere(1);Vrai;Vrai;Vrai)
```

FIXER LIENS AUTOMATIQUES

FIXER LIENS AUTOMATIQUES (aller {; retour})

Paramètre	Type		Description
aller	Booléen	⇒	Statut de tous les liens de N vers 1
retour	Booléen	⇒	Statut de tous les liens de 1 vers N

Description

La commande **FIXER LIENS AUTOMATIQUES** transforme tous les liens manuels en liens automatiques pour toute la base dans le process courant. Cette modification est temporaire et peut à tout moment être remise en cause par un nouvel appel à **FIXER LIENS AUTOMATIQUES**.

- Si *aller* est **Vrai**, tous les liens N vers 1 deviennent automatiques. Si *aller* est **Faux**, tous les liens N vers 1 deviennent manuels.
- Si *retour* est **Vrai**, tous les liens 1 vers N deviennent automatiques. Si *retour* est **Faux**, tous les liens 1 vers N deviennent manuels.

Les liens définis comme automatiques en mode Développement ne sont pas affectés par cette commande. Elle permet de rendre automatiques les liens déclarés manuels en mode Développement, avant d'exécuter des opérations nécessitant qu'ils soient automatiques (par exemple, des recherches et tri relationnels). A l'issue de l'opération, le lien peut redevenir manuel via un nouvel appel à **FIXER LIENS AUTOMATIQUES**.

Note : Lorsque vous passez **Vrai** à la commande **FIXER LIENS AUTOMATIQUES**, le mode automatique est "verrouillé" pour tous les liens manuels au cours de la session. Dans ce cas, les éventuels appels à la commande **FIXER LIEN CHAMP** dans la même session sont ignorés, qu'ils soient placés avant ou après **FIXER LIENS AUTOMATIQUES**. Pour "déverrouiller" le mode automatique et prendre en compte les appels à **FIXER LIEN CHAMP**, passez **Faux** à **FIXER LIENS AUTOMATIQUES**.

Exemple

L'exemple suivant rend tous les liens N vers 1 automatiques et rétablit en manuel tous les liens 1 vers N qui étaient précédemment modifiés :

```
FIXER LIENS AUTOMATIQUES(Vrai;Faux)
```

JOINTURE

JOINTURE (tableN ; table1)

Paramètre	Type		Description
tableN	Table	➡	Nom de la table N (d'où part le lien)
table1	Table	➡	Nom de la table 1 (où arrive le lien)

Description

La commande **JOINTURE** crée une nouvelle sélection d'enregistrements dans *table1* à partir de la sélection d'enregistrements de la *tableN* qui lui est liée et charge le premier enregistrement de la nouvelle sélection en tant qu'enregistrement courant.

Cette commande ne peut être utilisée que s'il existe un lien de **N** vers *1*. **JOINTURE** peut opérer au travers de plusieurs niveaux de liens. Il peut y avoir plusieurs tables liées entre la table **N** et la table *1*. Les liens peuvent être manuels ou automatiques.

JOINTURE utilise le chemin "le plus court" pour passer de la table de départ à la table d'arrivée. Si les chemins existants sont de taille équivalente, **JOINTURE** utilise le premier chemin trouvé dans l'ordre de création des champs de la table de départ.

Exemple

Nous souhaitons trouver tous les clients dont les factures arrivent à échéance aujourd'hui.
L'exemple suivant propose une méthode pour créer une sélection dans la table *[Clients]* à partir d'une sélection d'enregistrements de la table *[Factures]* :

```
ENSEMBLE VIDE([Clients];"Paiement Du")
CHERCHER([Factures]; [Factures]PaiementDu=Date du jour)
Tant que(Non(Fin de selection([Factures])))
   CHARGER SUR LIEN([Factures]ClientID)
   ADJOINDRE ELEMENT([Clients];"Paiement Du")
   ENREGISTREMENT SUIVANT([Factures])
Fin tant que
```

L'exemple suivant parvient au même résultat que le précédent :

```
CHERCHER([Factures];[Factures]PaiementDu=Date du jour)
JOINTURE([Factures];[Clients])
```

Note : Depuis la version 11, ce code peut également être écrit de la manière suivante sans perte de performances :

```
CHERCHER([Clients];[Factures]PaiementDu=Date du jour)
```

LIEN RETOUR

LIEN RETOUR (table1 | champ1)

Paramètre	Type		Description
table1 | champ1	Table, Champ	➡	Table pour laquelle établir tous les liens de 1 vers N ou champ 1

Description

LIEN RETOUR a deux syntaxes.

La première syntaxe, **LIEN RETOUR(table1)**, active tous les liens 1 vers N pour *table1*. Elle modifie la sélection courante pour chaque table qui a un lien 1 vers N vers *table1*. Les sélections courantes dans les tables N dépendent de la valeur courante de chaque champ lié dans la table 1. Chaque fois que cette commande est exécutée, les sélections courantes des tables N sont modifiées et le premier enregistrement de la sélection est chargé en tant qu'enregistrement courant.

La seconde syntaxe, **LIEN RETOUR(champ1)**, active le lien 1 vers N pour *champ1*. Elle modifie la sélection courante et l'enregistrement courant pour chaque table qui a un lien avec *champ1*. En conséquence, les enregistrements liés deviennent la sélection courante de la table N.

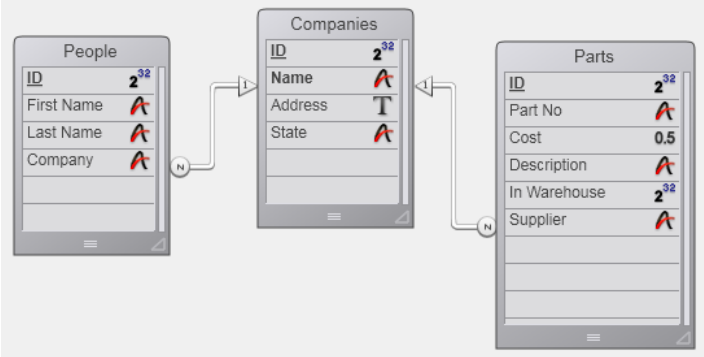
Notes :

- Si la sélection courante de la table 1 est vide au moment de l'exécution de **LIEN RETOUR**, la commande ne fait rien.
- Pour que la commande fonctionne, les champs clé d'appel (champs N) doivent être indexés.

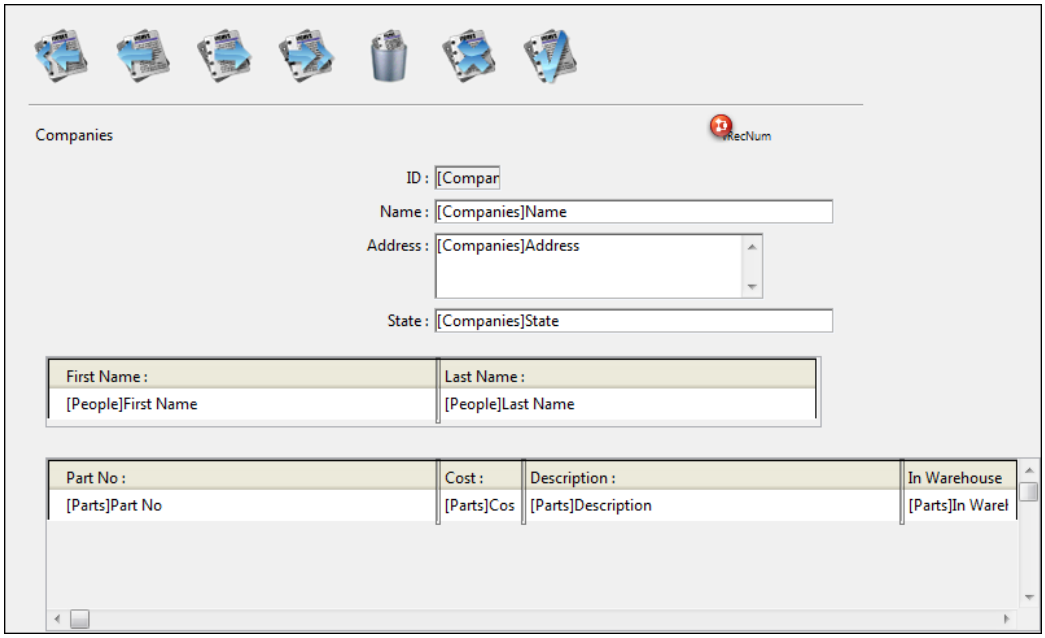
Note : Cette commande ne prend pas en charge les champs de type Objet.

Exemple

Dans l'exemple suivant, trois tables sont liées avec des liens automatiques. Les deux tables [People] et [Parts] ont un lien N vers 1 vers la table [Companies].



Voici le formulaire pour la table [Companies] qui affiche les enregistrements liés venant des tables [People] et [Parts].



Lorsque les formulaires pour People et Parts s'affichent, les enregistrements liés pour les tables [People] et [Parts] sont chargés et deviennent les sélections courantes de ces tables.

En revanche, les enregistrements liés ne sont pas chargés si un enregistrement de la table [Companies] est sélectionné par programmation. Dans ce cas, il faut utiliser la commande **LIEN RETOUR**.

Par exemple, la méthode suivante effectue une boucle sur chaque enregistrement de la table [Companies]. Pour chaque société, une alerte apparaît. Cette alerte affiche le nombre de personnes dans la société (le nombre d'enregistrements liés dans la table [People]) ainsi que le nombre de Parts que la société distribue (le nombre d'enregistrements dans la table [Parts] qui sont liés). Notez que nous avons besoin d'appeler la commande **LIEN RETOUR** bien que

les liens soient automatiques :

```
 TOUT SELECTIONNER([Companies]) ` Sélectionner tous les enregistrements dans la table
 TRIER([Companies];[Companies]Nom) ` Trier les enregistrements dans l'ordre alphabétique
 Boucle($i;1;Enregistrements dans table([Companies])) ` Boucler une fois par enregistrement
    LIEN RETOUR([Companies]Nom) ` Sélectionner les enregistrements liés
    ALERTE("Société : "+[Companies]Nom+Caractere(13)+"personnes dans la société : "+Chaine(Enregistrements
trouves([People]))+Caractere(13)+"Nombre de Produits qu'ils distribuent : "+Chaine(Enregistrements
trouves([Parts])))
    ENREGISTREMENT SUIVANT([Companies]) ` Aller à l'enregistrement suivant
 Fin de boucle
```

LIRE LIEN CHAMP

LIRE LIEN CHAMP (champN ; aller ; retour {; *})

Paramètre	Type		Description
champN	Champ	⇨	Champ de départ du lien
aller	Entier long	⇦	Statut du lien aller
retour	Entier long	⇦	Statut du lien retour
*	Opérateur	⇨	• Si passé : aller et retour retournent le statut courant effectif du lien (valeurs 2 ou 3 uniquement) • Si omis (défaut) : aller et retour peuvent retourner la valeur 1 si le lien n'a pas été modifié par programmation

Description

La commande **LIRE LIEN CHAMP** permet de connaître le statut automatique/manuel du lien partant du *champN* pour le process courant. Tous les liens peuvent être consultés, y compris les liens déclarés automatiques dans la fenêtre de Structure.

- Passez dans *champN* le nom du champ de la table N d'où part le lien dont vous souhaitez connaître le statut. Si aucun lien ne part du champ *champN*, les paramètres *aller* et *retour* retournent 0, une erreur est générée et la variable système OK prend la valeur 0 (cf. ci-dessous).
- Après l'exécution de la commande, la variable *aller* contient une valeur indiquant si le lien aller spécifié est défini comme automatique :
  0 = il n'y a pas de lien partant de *champN*. L'erreur de syntaxe n°16 ("Ce champ ne possède pas de lien") est générée et la variable système OK prend la valeur 0.
  1 = le statut automatique/manuel du lien aller spécifié est celui défini par l'option **Lien aller auto** dans les propriétés du lien en mode Développement (il n'a pas été modifié par programmation).
  2 = le lien N vers 1 est manuel pour le process.
  3 = le lien N vers 1 est automatique pour le process.
- Après l'exécution de la commande, la variable *retour* contient une valeur indiquant si le lien retour spécifié est défini comme automatique :
  0 = il n'y a pas de lien partant de *champN*. L'erreur de syntaxe n°16 ("Ce champ ne possède pas de lien") est générée et la variable système OK prend la valeur 0.
  1 = le statut automatique/manuel du lien retour spécifié est celui défini par l'option **Lien retour auto** dans les propriétés du lien en mode Développement (il n'a pas été modifié par programmation).
  2 = le lien 1 vers N est manuel pour le process.
  3 = le lien 1 vers N est automatique pour le process.

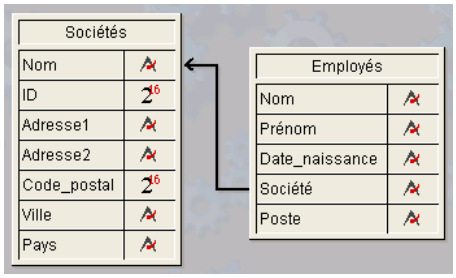
Vous pouvez comparer les valeurs reçues dans les paramètres *aller* et *retour* aux constantes du thème "**Liens**" :

Constante	Type	Valeur
Automatique	Entier long	3
Configuration structure	Entier long	1
Manuel	Entier long	2
Pas de lien	Entier long	0

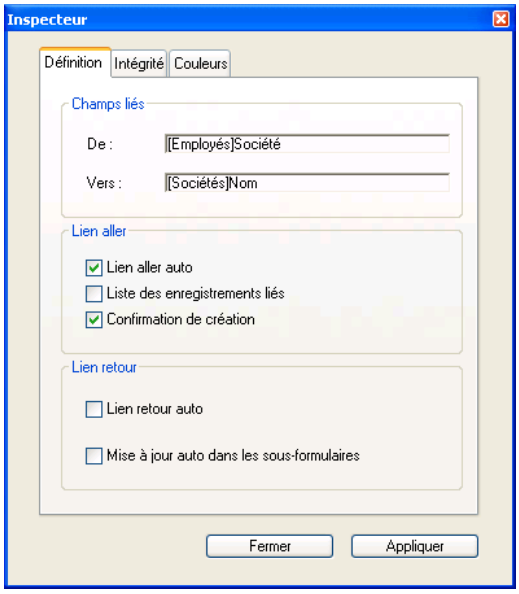
- Le paramètre facultatif * permet de "forcer" la lecture du statut courant du lien, même s'il n'a pas été modifié par programmation. Autrement dit, lorsque vous passez le paramètre *, seules les valeurs 2 ou 3 peuvent être retournées dans les paramètres *aller* et *retour*.

Exemple

Soit la structure suivante :



Les propriétés du lien reliant le champ [Employés]Société au champ [Sociétés]Nom sont les suivantes :



Le code ci-dessous illustre les différentes possibilités offertes par les commandes **LIRE LIEN CHAMP**, **LIRE LIENS AUTOMATIQUES** et **FIXER LIEN CHAMP**, **FIXER LIENS AUTOMATIQUES** ainsi que leurs effets :

```
LIRE LIENS AUTOMATIQUES(liens_Appel;liens_Retour) `retourne Faux, Faux
LIRE LIEN CHAMP([Employés]Société;aller;retour) `retourne 1,1
LIRE LIEN CHAMP([Employés]Société;aller;retour;*) `retourne 3,2

FIXER LIEN CHAMP([Employés]Société;2;0) `passe le lien N vers 1 en manuel

LIRE LIEN CHAMP([Employés]Société;aller;retour) `retourne 2,1
LIRE LIEN CHAMP([Employés]Société;aller;retour;*) `retourne 2, 2

FIXER LIEN CHAMP([Employés]Société;1;0) `rétablit les paramètres définis en
 `structure pour le lien N vers 1

LIRE LIEN CHAMP([Employés]Société;aller;retour) `retourne 1,1
LIRE LIEN CHAMP([Employés]Société;aller;retour;*) `retourne 3,2

FIXER LIENS AUTOMATIQUES(Vrai;Vrai) `passe tous les liens de toutes les tables en automatique

LIRE LIENS AUTOMATIQUES(liens_Appel;liens_Retour) `retourne Vrai, Vrai
LIRE LIEN CHAMP([Employés]Société;aller;retour) `retourne 1,1
LIRE LIEN CHAMP([Employés]Société;aller;retour;*) `retourne 3,3
```

LIRE LIENS AUTOMATIQUES

LIRE LIENS AUTOMATIQUES (aller ; retour)

Paramètre	Type		Description
aller	Booléen	⇐	Statut de tous les liens de N vers 1
retour	Booléen	⇐	Statut de tous les liens de 1 vers N

Description

La commande **LIRE LIENS AUTOMATIQUES** permet de savoir si le statut automatique/manuel de tous les liens manuels N vers 1 et 1 vers N de la base a été modifié dans le process courant.

- *aller* : ce paramètre retourne **Vrai** si un appel antérieur de la commande **FIXER LIENS AUTOMATIQUES** a rendu automatiques tous les liens manuels N vers 1 — par exemple **FIXER LIENS AUTOMATIQUES(Vrai;Faux)**.
  Ce paramètre retourne **Faux** si la commande **FIXER LIENS AUTOMATIQUES** n'a pas été appelée ou si sa précédente exécution n'a pas modifié les liens manuels N vers 1 — par exemple **FIXER LIENS AUTOMATIQUES(Faux;Faux)**.
- *retour* : ce paramètre retourne **Vrai** si l'appel précédent de la commande **FIXER LIENS AUTOMATIQUES** a rendu automatiques tous les liens manuels 1 vers N — par exemple **FIXER LIENS AUTOMATIQUES(Vrai;Vrai)**.
  Ce paramètre retourne **Faux** si la commande **FIXER LIENS AUTOMATIQUES** n'a pas été appelée ou si sa précédente exécution n'a pas modifié les liens manuels 1 vers N — par exemple **FIXER LIENS AUTOMATIQUES(Vrai;Faux)**.

Exemple

Reportez-vous à l'exemple de la commande **LIRE LIEN CHAMP**.

SELECTION RETOUR

SELECTION RETOUR (leChamp)

Paramètre	Type		Description
leChamp	Champ	⇨	Champ de la table N (d'où part le lien)

Description

La commande **SELECTION RETOUR** crée une sélection d'enregistrements dans la table N basée sur la sélection courante de la table 1, et charge le premier enregistrement de la table N comme enregistrement courant.

Note : SELECTION RETOUR modifie l'enregistrement courant de la table 1.

Exemple

Prenons l'exemple d'une base de données comportant une table *[Factures]* dont le champ *[Factures]IDClient* est lié au champ *[Clients]NoID* de la table *[Clients]*. L'exemple suivant sélectionne toutes les factures adressées aux clients dont le crédit est supérieur ou égal à 5710 Euros :

```
  ` Sélectionner les clients
CHERCHER([Clients];[Clients]Credit>=5710)
 `Trouver toutes les factures liées à chacun de ces clients
SELECTION RETOUR([Factures]IDClient)
```

STOCKER SUR LIEN

STOCKER SUR LIEN (leChamp)

Paramètre	Type		Description
leChamp	Champ	⇨	Champ N

Description

STOCKER SUR LIEN sauvegarde l'enregistrement lié à *leChamp*. Vous pouvez exécuter une commande **STOCKER SUR LIEN** pour mettre à jour un enregistrement créé par **CREER SUR LIEN**, ou bien lorsque vous voulez sauvegarder des modifications apportées à un enregistrement chargé par **CHARGER SUR LIEN**.

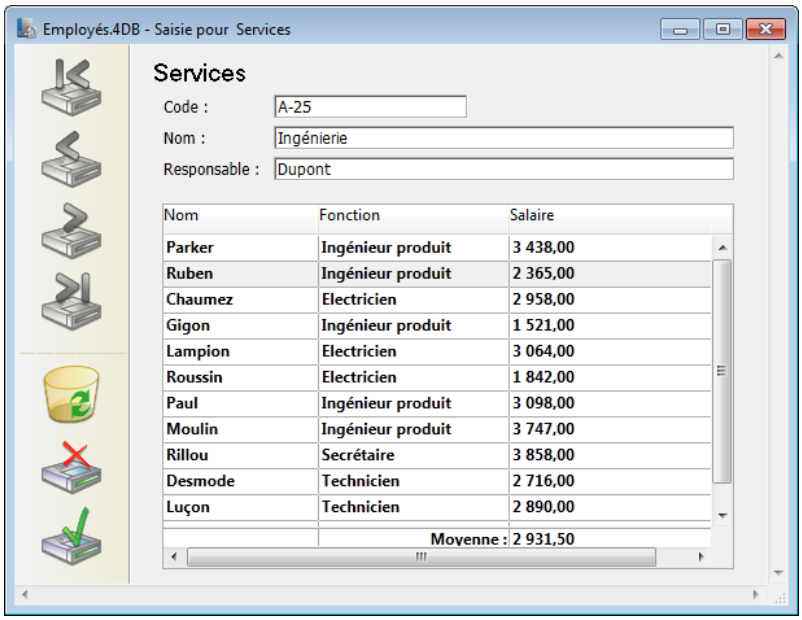
STOCKER SUR LIEN ne sauvegardera pas un enregistrement verrouillé. Lorsque vous appelez cette commande, vous devez tout d'abord vous assurer que l'enregistrement n'est pas verrouillé. S'il est verrouillé, la commande est ignorée, l'enregistrement n'est pas sauvegardé et aucune erreur ne vous est retournée.

List Box

- Gestion programmée des objets de type List box
- Gestion des List box hiérarchiques
- Utiliser des tableaux objets dans les colonnes (4D View Pro)
- 4D View Pro
- LISTBOX CONTRACTER
- LISTBOX DEPLACER COLONNE
- LISTBOX DEPLOYER
- LISTBOX DUPLIQUER COLONNE
- LISTBOX FIXER CALCUL PIED
- LISTBOX FIXER COLONNES STATIQUES
- LISTBOX FIXER COLONNES VERROUILLEES
- LISTBOX FIXER COULEUR GRILLE
- LISTBOX FIXER COULEUR LIGNE
- LISTBOX FIXER FORMULE COLONNE
- LISTBOX FIXER GRILLE
- LISTBOX FIXER HAUTEUR ENTETES
- LISTBOX FIXER HAUTEUR LIGNE Nouveauté 16.0
- LISTBOX FIXER HAUTEUR LIGNES
- LISTBOX FIXER HAUTEUR PIEDS
- LISTBOX FIXER HIERARCHIE
- LISTBOX FIXER LARGEUR COLONNE
- LISTBOX FIXER STYLE LIGNE
- LISTBOX FIXER TABLE SOURCE
- LISTBOX FIXER TABLEAU Modifié 16.0
- LISTBOX INSERER COLONNE
- LISTBOX INSERER COLONNE FORMULE
- LISTBOX INSERER LIGNES
- LISTBOX Lire calcul pied
- LISTBOX Lire colonnes statiques
- LISTBOX Lire colonnes verrouillees
- LISTBOX LIRE COORDONNEES CELLULE
- LISTBOX LIRE COULEUR GRILLE
- LISTBOX Lire couleur ligne
- LISTBOX Lire formule colonne
- LISTBOX LIRE GRILLE
- LISTBOX Lire hauteur entetes
- LISTBOX Lire hauteur ligne Nouveauté 16.0
- LISTBOX Lire hauteur lignes
- LISTBOX Lire hauteur pieds
- LISTBOX LIRE HIERARCHIE
- LISTBOX Lire information
- LISTBOX LIRE INFORMATION IMPRESSION
- LISTBOX Lire largeur colonne
- LISTBOX Lire nombre colonnes
- LISTBOX Lire nombre lignes
- LISTBOX LIRE OBJETS
- LISTBOX LIRE POSITION CELLULE
- LISTBOX Lire style ligne
- LISTBOX LIRE TABLE SOURCE
- LISTBOX Lire tableau Modifié 16.0
- LISTBOX LIRE TABLEAUX
- LISTBOX NUMERO COLONNE DEPLACEE
- LISTBOX NUMERO LIGNE DEPLACEE
- LISTBOX SELECTIONNER LIGNE
- LISTBOX SELECTIONNER RUPTURE
- LISTBOX SUPPRIMER COLONNE
- LISTBOX SUPPRIMER LIGNES
- LISTBOX TRIER COLONNES

Gestion programmée des objets de type List box

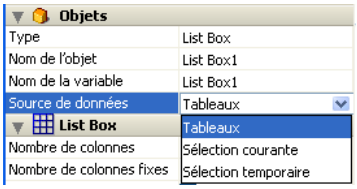
Les commandes de ce thème sont dédiées à la gestion programmée des objets de formulaire de type List box.
Les list box permettent de représenter des données sous forme de colonnes et de lignes sélectionnables et proposent de nombreuses fonctions d'interface telles que la possibilité de saisir des valeurs, trier les colonnes, afficher une hiérarchie, définir des couleurs alternées, etc.



Un objet de type List box est entièrement paramétrable dans l'éditeur de formulaires de 4D et peut également être contrôlé par programmation. Pour plus d'informations sur la création et le paramétrage des objets de type List box dans l'éditeur de formulaires ainsi que leur utilisation, reportez-vous au manuel *Mode Développement* de la documentation de 4D.
La programmation des objets de type List box s'effectue sur le même modèle que les autres objets de formulaire en liste de 4D. Elle doit cependant tenir compte de principes spécifiques, décrits dans cette section.

Sources de données et principes de gestion des valeurs

Un objet List box peut contenir une ou plusieurs colonnes et peut être associé soit à des tableaux 4D, soit à une sélection d'enregistrements. Dans le cas des list box de type sélection, les colonnes sont associées à des champs ou des expressions.
Il n'est pas possible de combiner dans une même list box ces deux types de sources de données (tableaux et sélections). La définition de la source de données s'effectue au moment de la création de l'objet List box dans l'éditeur de formulaires, via la Liste des propriétés. Il n'est pas possible de la modifier ensuite par programmation.



List box de type tableau

Dans ce type de list box, chaque colonne est associée à un tableau 4D à une dimension ; tous les types de tableaux peuvent être utilisés, à l'exception des tableaux de pointeurs. Le format d'affichage de chaque colonne peut être défini dans l'éditeur de formulaires ou via la commande **OBJET FIXER FORMATAGE**.

En mode programmé, les valeurs des colonnes (saisie et affichage) sont gérées à l'aide des commandes de haut niveau du thème List box (telles que **LISTBOX INSERER LIGNES** ou **LISTBOX INSERER COLONNE**) ainsi que des commandes de manipulation des tableaux.
Par exemple, pour initialiser le contenu d'une colonne de List box, vous pouvez utiliser l'instruction suivante :

```
TABLEAU TEXTE(NomColonne;taille)
```

Vous pouvez également utiliser une énumération :

```
LISTE VERS TABLEAU("NomEnum";NomColonne)
```

Attention : Lorsqu'un objet List box contient plusieurs colonnes de tailles différentes, seul le nombre d'éléments correspondant au plus petit tableau est affiché. Il est donc conseillé de veiller à ce que chaque tableau ait le même nombre d'éléments que les autres. A noter également que si une colonne de la list box est "vide" (c'est le cas lorsque le tableau associé n'a pas été déclaré ou dimensionné via le langage), la list box n'affiche aucun contenu.

List box de type sélection

Dans ce type de list box, chaque colonne peut être associée à un champ ou à une expression. Le contenu de chaque ligne est alors évalué en fonction d'une sélection d'enregistrements : la sélection courante d'une table ou une sélection temporaire.
Dans le cas de la sélection courante, toute modification effectuée côté base de données est automatiquement reportée dans la list box et inversement. La sélection courante est donc toujours identique aux deux emplacements. A noter que les commandes **LISTBOX INSERER LIGNES** et **LISTBOX SUPPRIMER LIGNES** ne peuvent pas être utilisées avec les list box de type sélection.

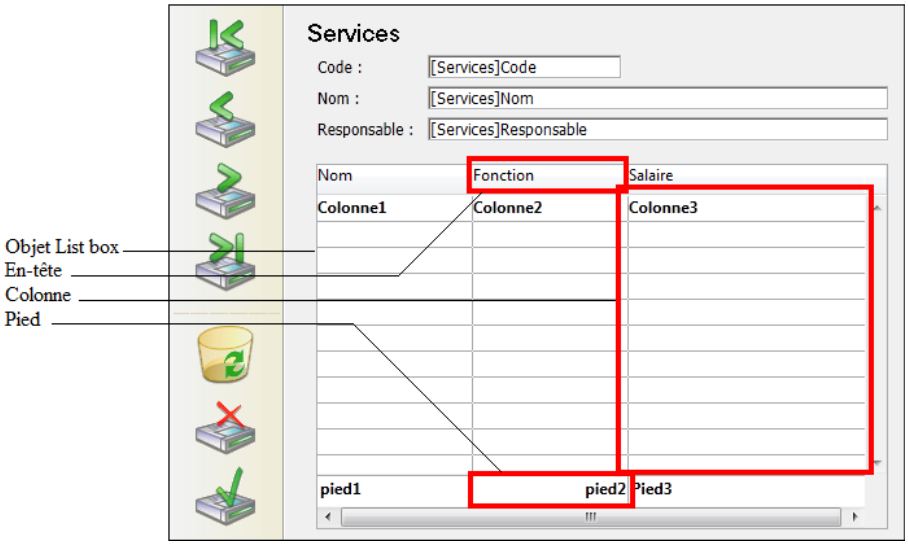
Vous pouvez associer une colonne de list box à une expression. L'expression pourra être basée sur un ou plusieurs champs (par exemple *[Employés]Nom+" "+[Employés]Prénom*) ou être simplement une formule (par exemple **Chaine**(**Nombre de millisecondes**)). L'expression peut également être une méthode projet (devant retourner une valeur dans $0), une variable ou un élément de tableau.
La commande **LISTBOX FIXER TABLE SOURCE** permet de modifier par programmation la table associée à la list box.

Objet, colonne, en-tête et pied

Un objet List box est composé de quatre types d'éléments distincts :

- l'**objet** dans son ensemble,
- les **colonnes**,
- les **en-têtes** des colonnes (peuvent être affichés ou masqués, sont affichés par défaut),
- les **pieds** des colonnes (peuvent être affichés ou masqués, sont masqués par défaut).

Dans l'éditeur de formulaires, ces éléments peuvent être sélectionnés séparément. Chacun d'eux dispose de son propre nom d'objet et nom de variable et peut donc être adressé séparément.



Par défaut, les colonnes sont nommées **Colonne1** à *n*, les en-têtes **Entête1** à *n* et les pieds **Pied1** à *n* dans le formulaire, indépendamment des objets List box eux-mêmes. A noter que, par défaut, le même nom est utilisé pour les objets et leurs variables associées, à l'exception des pieds (par défaut les variables sont vides pour les pieds, 4D utilise des variables dynamiques).
Chaque type d'élément dispose de caractéristiques propres et de caractéristiques partagées avec les autres éléments. Par exemple, la police de caractères peut être assignée globalement à l'objet List box ou séparément aux colonnes, aux en-têtes et/ou aux pieds. A l'inverse, les propriétés de saisie ne sont définissables que pour les colonnes.
Ces principes s'appliquent aux commandes du thème **Objets de formulaires** pouvant être utilisées avec les list box : en fonction de sa nature, chaque commande sera utilisable avec la list box, les colonnes, les en-têtes et/ou les pieds des colonnes. Pour désigner le type d'élément sur lequel vous souhaitez agir, il suffit de passer le nom d'objet ou la variable qui lui est associé(e).
Le tableau suivant précise la portée de chaque commande du thème **Objets de formulaires** utilisable avec les objets de type list box :

Commandes Propriétés des objets	Objet	Colonne	En-tête de colonne	Pied de colonne
OBJET DEPLACER	X			
OBJET LIRE COORDONNEES	X	X	X	X
OBJET FIXER REDIMENSIONNEMENT	X			
OBJET LIRE REDIMENSIONNEMENT	X			
OBJET LIRE TAILLE OPTIMALE		X		
OBJET FIXER FILTRE SAISIE		X		
OBJET FIXER FORMATAGE		X	X	X
OBJET FIXER SAISISSABLE		X		
OBJET FIXER LISTE PAR NOM		X		
OBJET FIXER TITRE			X	
OBJET FIXER COULEUR	X	X	X	X
OBJET FIXER COULEURS RVB	X	X	X	X
OBJET FIXER POLICE	X	X	X	X
OBJET FIXER TAILLE POLICE	X	X	X	X
OBJET FIXER STYLE POLICE	X	X	X	X
OBJET FIXER ALIGNEMENT HORIZONTAL	X	X	X	X
OBJET Lire alignement horizontal	X	X	X	X
OBJET FIXER ALIGNEMENT VERTICAL	X	X	X	X
OBJET Lire alignement vertical	X	X	X	X
OBJET FIXER VISIBLE	X	X	X	X
OBJET FIXER BARRES DEFILEMENT	X			
OBJET FIXER DEFILEMENT	X			

Note : Avec les List box de type tableau, il est possible de définir séparément les propriétés de style, de couleur de police, de couleur de fond et de visibilité de chaque ligne. Cette gestion s'effectue via des tableaux associés à la list box dans la Liste des propriétés. Vous pouvez récupérer par programmation le nom de ces tableaux à l'aide de la commande **LISTBOX LIRE TABLEAUX**.

List box et Langage

Méthodes objet

Il est possible d'associer une méthode à l'objet List box dans son ensemble et/ou à chaque colonne de la list box. Les méthodes objet sont appelées dans l'ordre suivant :

1. Méthode objet de la colonne

2. Méthode objet de la list box

La méthode objet de la colonne reçoit les événements se produisant dans son en-tête et dans son pied.

OBJET FIXER VISIBLE et en-têtes/pieds

Lorsque la commande **OBJET FIXER VISIBLE** est utilisée avec un en-tête ou un pied de list box, elle agit sur tous les en-têtes ou tous les pieds de l'objet List box, quel que soit l'élément individuel spécifié par la commande. Par exemple, l'instruction **OBJET FIXER VISIBLE(*;"entête3";Faux)** masquera tous les en-têtes de l'objet List box auquel appartient l'en-tête 3 (et non uniquement cet en-tête).
A noter que, pour que vous puissiez gérer la visibilité de ces objets à l'aide de la commande **OBJET FIXER VISIBLE**, ils doivent avoir été affichés dans la list box au niveau de l'éditeur de formulaires (l'option **Afficher en-têtes** et/ou **Afficher pieds** doit être cochée pour l'objet).

Objet Lire pointeur

La fonction **OBJET Lire pointeur** utilisée avec la constante Objet avec focus ou Objet courant (anciennement fonctions **Objet focus** et **Self**) peut être utilisée dans la méthode objet d'une list box ou d'une colonne de list box. Elle retourne un pointeur vers la list box, la colonne de list box (1) ou la variable d'en-tête en fonction du type d'événement formulaire. Le tableau suivant détaille ce fonctionnement :

Evénement	Objet avec focus	Objet courant
Sur clic	list box	colonne
Sur double clic	list box	colonne
Sur avant frappe clavier	colonne	colonne
Sur après frappe clavier	colonne	colonne
Sur après modification	colonne	colonne
Sur gain focus	colonne ou list box (*)	colonne ou list box (*)
Sur perte focus	colonne ou list box (*)	colonne ou list box (*)
Sur déposer	list box source	list box (*)
Sur glisser	list box source	list box (*)
Sur début glisser	list box	list box (*)
Sur début survol	list box (**)	list box (**)
Sur survol	list box (**)	list box (**)
Sur fin survol	list box (**)	list box (**)
Sur données modifiées	colonne	colonne
Sur nouvelle sélection	list box (**)	list box (**)
Sur avant saisie	colonne	colonne
Sur déplacement colonnee	list box	colonne
Sur déplacement ligne	list box	list box
Sur redimensionnement colonne	list box	colonne
Sur ouverture corps	Nil	list box (**)
Sur fermeture corps	Nil	list box (**)
Sur clic entête	list box	en-tête
Sur clic pied	list box	pied
Sur après tri	list box	en-tête

(*) Lorsque le focus est modifié à l'intérieur d'une list box, un pointeur vers la colonne est retourné. Lorsque le focus est modifié au niveau global du formulaire, un pointeur vers la list box est retourné. Dans le contexte d'une méthode objet de colonne, un pointeur vers la colonne est retourné.
(**) Non exécuté dans le contexte d'une méthode objet de colonne.

(1) Lorsqu'un pointeur vers la colonne est retourné, l'objet pointé dépend du type de la list box. Dans le cadre d'une list box de type tableau, **OBJET Lire pointeur** retourne un pointeur vers le tableau. Le mécanisme des pointeurs de 4D permet alors de connaître le numéro de l'élément de tableau modifié. Par exemple, en supposant que l'utilisateur a modifié la 5e ligne de la colonne col2 :

```
$Colonne:=OBJET Lire pointeur(Objet avec focus)
  ` $Colonne contient un pointeur vers col2
$Ligne:=$Colonne-> `$Ligne vaut 5
```

Dans le cadre d'une list box de type sélection, **OBJET Lire pointeur** retourne :

- pour une colonne associée à un champ, un pointeur vers le champ associé,
- pour une colonne associée à une variable, un pointeur vers la variable,
- pour une colonne associée à une expression, un pointeur **Nil**.

OBJET FIXER DEFILEMENT

Il est possible d'utiliser la commande **OBJET FIXER DEFILEMENT** (thème "**Objets (Formulaires)**") avec un objet de type list box. Cette commande permet de faire défiler les lignes de la list box afin d'afficher la première ligne sélectionnée ou une ligne spécifique.

EDITER ELEMENT

La commande **EDITER ELEMENT** (thème "**Gestion de la saisie**") permet de passer en mode édition une cellule d'un objet list box.

REDESSINER

Lorsqu'elle est appliquée à une list box en mode sélection, la commande **REDESSINER** (thème "**Interface utilisateur**") provoque la mise à jour des données affichées dans la list box.

Numero de ligne affichee

La commande **Numero de ligne affichee** (thème "**Sélections**") fonctionne dans le contexte de l'événement Sur affichage corps pour un objet list box.

Evénements formulaire

Des événements formulaire spécifiques sont destinés à la gestion programmée des list box, concernant notamment le glisser-déposer et le tri. Pour plus d'informations, reportez-vous à la description de la commande **Evenement formulaire**.

Glisser déposer

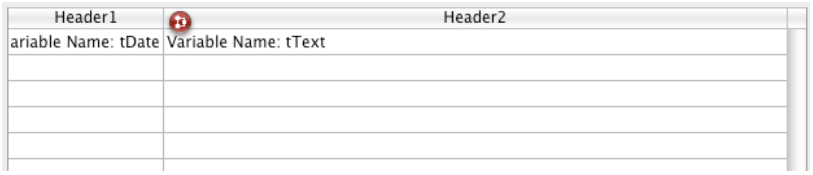
La gestion du glisser-déposer de données dans les list box est prise en charge par les commandes **Position deposer** et **PROPRIETES GLISSER DEPOSER**. Ces commandes ont été spécialement adaptées pour les list box.
Attention de ne pas confondre le glisser-déposer avec le déplacement de lignes et de colonnes, pris en charge par les commandes **LISTBOX NUMERO LIGNE DEPLACEE** et **LISTBOX NUMERO COLONNE DEPLACEE**.

Gestion de la saisie

Pour qu'une cellule de list box soit saisissable, quel que soit son type (tableau et sélection), il est nécessaire que les deux conditions suivantes soient réunies :

- la colonne de la cellule a été définie comme **Saisissable** (dans le cas contraire, les cellules de la colonne ne seront jamais saisissables).
- dans l'événement formulaire Sur avant saisie, $0 ne retourne pas -1.
  Lorsque le curseur arrive dans la cellule, l'événement Sur avant saisie est généré dans la méthode de la colonne. Si, dans le contexte de cet événement, $0 prend la valeur -1, la cellule est considérée comme non saisissable. Si l'événement a été généré suite à un appui sur **Tabulation** ou **Maj+Tabulation**, le focus est donné à la cellule respectivement suivante ou précédente. Si $0 ne vaut pas -1 (par défaut $0 vaut 0), la cellule est saisissable et passe bien en édition.

Imaginons par exemple une list box contenant deux tableaux, de type date et texte. Le tableau date n'est pas saisissable. Le tableau texte est saisissable si la date n'est pas déjà passée.



Voici la méthode de la colonne tText :

```
Au cas ou
   :(Evenement formulaire=Sur avant saisie) // une cellule prend le focus
      LISTBOX LIRE POSITION CELLULE(*;"lb";$col;$row)
 // identification de la cellule
      Si(tDate{$row}<Date du jour) // si la date est antérieure à aujourd'hui
         $0:=-1 // la cellule n'est PAS saisissable
      Sinon
         ... // la cellule est saisissable
      Fin de si
Fin de cas
```

Note : A compter de 4D v13, l'événement Sur avant saisie est retourné avant Sur gain focus.

A noter qu'avec les list box de type sélection, afin de préserver la cohérence des données, tout enregistrement modifié est sauvegardé dès qu'une cellule est validée (le trigger Sur sauvegarde enregistrement est appelé s'il est défini) et l'événement Sur données modifiées n'est exécuté qu'ensuite. La séquence type d'événements générée lors de la saisie ou de la modification de données est la suivante :

Action	Séquence d'événements
Une cellule passe en édition	Sur avant saisie / Sur gain focus
La valeur est modifiée	Sur avant frappe clavier / Sur après frappe clavier / Sur après modification
L'utilisateur valide et quitte la cellule (tabulation, clic...)	*Sauvegarde* (Trigger Sur sauvegarde enregistrement) / Sur données modifiées / Sur perte focus

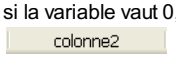
Gestion des tris

Par défaut, la list box gère automatiquement les tris standard des colonnes en cas de clic sur l'en-tête. Un tri standard est un tri alphanumérique des valeurs de la colonne, alternativement croissant / décroissant lors de clics multiples. Toutes les colonnes sont toujours automatiquement synchronisées.
Il est possible d'interdire le tri utilisateur standard en désélectionnant la propriété "Triable" pour la list box.

Le développeur peut mettre en place une gestion personnalisée des tris à l'aide de la commande **LISTBOX TRIER COLONNES** et/ou en combinant les événements formulaire Sur clic entête et Sur après tri (cf. commande **Evenement formulaire**) et les commandes 4D de gestion des tableaux.

Note : La propriété "Triable" concerne uniquement le tri utilisateur standard, la commande **LISTBOX TRIER COLONNES** ne tient pas compte de cette propriété.

En outre, la valeur de la variable associée à l'en-tête d'une colonne permet de gérer une information supplémentaire : le tri courant de la colonne (lecture) et l'affichage de la flèche de tri.

- si la variable vaut 0, la colonne n'est pas triée et la flèche de tri n'est pas affichée ;
  

- si la variable vaut 1, la colonne est triée par ordre croissant et la flèche de tri croissant est affichée ;

- si la variable vaut 2, la colonne est triée par ordre décroissant et la flèche de tri décroissant est affichée.
  

Il est possible de fixer la valeur de la variable (par exemple **Header2:=2**) afin de "forcer" l'affichage de la flèche de tri. Le tri de la colonne lui-même n'est dans ce cas pas modifié, il appartient au développeur de le gérer.

Note : La commande **OBJET FIXER FORMATAGE** propose une prise en charge spécifique pour les icônes dans les en-têtes de list box, utile lorsque vous voulez travailler avec une icône de tri personnalisée.

Gestion des sélections

La gestion des sélections s'effectue différemment pour les list box de type tableau et de type sélection.

- List box de type sélection : les sélections sont gérées par l'intermédiaire d'un ensemble appelé par défaut *$ListboxSetN* (*N* débute à 0 et est incrémenté en fonction du nombre de list box dans le formulaire), que vous pouvez modifier si nécessaire. Cet ensemble est défini dans les propriétés de la list box. Il est maintenu automatiquement par 4D : si l'utilisateur sélectionne une ou plusieurs ligne(s) dans la list box, l'ensemble est immédiatement mis à jour). A l'inverse, il est possible d'utiliser les commandes du thème "Ensembles" afin de modifier par programmation la sélection dans la list box.

- List box de type tableau : la commande **LISTBOX SELECTIONNER LIGNE** permet de sélectionner par programmation une ou plusieurs lignes de list box.
  En outre, la variable associée à l'objet List box peut être utilisée pour lire, fixer ou stocker les sélections de lignes dans l'objet. Cette variable correspond à un tableau de booléens automatiquement créé et maintenu par 4D. La taille de ce tableau est déterminée par celle de la list box : il contient le même nombre d'éléments que le plus petit tableau associé aux colonnes.
  Chaque élément de ce tableau contient **Vrai** si la ligne correspondante est sélectionnée et **Faux** sinon. 4D met à jour le contenu de ce tableau en fonction des actions utilisateur. A l'inverse, vous pouvez modifier la valeur des éléments de ce tableau afin de modifier la sélection dans la list box. En revanche, vous ne pouvez ni insérer ni supprimer de ligne dans ce tableau ; il n'est pas possible non plus de le retyper.

Note : La commande **Compter dans tableau** est utile dans ce cas pour connaître le nombre de lignes sélectionnées.

Par exemple, cette méthode permet d'inverser la sélection de la première ligne de la list box (type tableau) :

```
TABLEAU BOOLEEN(tBListBox;10)
  ` tBListBox est le nom de la variable associée à la List box dans le formulaire
Si(tBListBox{1}=Vrai)
   tBListBox{1}:=Faux
Sinon
   tBListBox{1}:=Vrai
Fin de si
```

Note : Les spécificités de la gestion des sélections dans les list box en mode hiérarchique sont détaillées dans la section **Gestion des List box hiérarchiques**.

Gestion des impressions

Il est possible d'imprimer des list box à compter de 4D v12. Deux modes d'impression sont proposés : le mode prévisualisation, permettant d'imprimer une list box comme un objet de formulaire et le mode avancé, permettant de contrôler l'impression de l'objet list box lui-même au sein du formulaire. A noter que l'apparence "Impression" est proposée pour les list box dans l'éditeur de formulaires.

Mode prévisualisation

L'impression d'une list box en mode prévisualisation consiste à imprimer directement la list box avec le formulaire qui la contient via les commandes d'impression standard ou la commande de menu **Imprimer**. La list box est imprimée dans l'état où elle se trouve dans le formulaire. Ce mode ne permet pas de contrôler précisément l'impression de l'objet ; en particulier, il ne permet pas d'imprimer toutes les lignes d'une list box contenant plus de lignes qu'elle ne peut en afficher.

Mode avancé

Dans ce mode, l'impression des list box s'effectue par programmation, via la commande **Imprimer objet** (les formulaires projet et les formulaires table sont pris en charge). La commande **LISTBOX LIRE INFORMATION IMPRESSION** permet de contrôler l'impression de l'objet.

Dans ce mode :

- la hauteur de l'objet list box est automatiquement réduite lorsque le nombre de lignes à imprimer est inférieur à la hauteur d'origine de l'objet (il n'y a pas de lignes "vides" imprimées). En revanche, la hauteur n'augmente pas automatiquement en fonction du contenu de l'objet. La taille de l'objet effectivement imprimé peut être obtenue via la commande **LISTBOX LIRE INFORMATION IMPRESSION**.
- l'objet list box est imprimé "tel quel", c'est-à-dire en tenant compte de ses paramètres d'affichage courants : visibilité des en-têtes et des grilles, lignes affichées et masquées, etc. Ces paramètres incluent également la première ligne à imprimer : si vous appelez la commande **OBJET FIXER DEFILEMENT** avant de lancer l'impression, la première ligne imprimée dans la list box sera celle désignée par la commande.
- un mécanisme automatique facilite l'impression des list box contenant plus de lignes qu'il est possible d'en afficher : des appels successifs à **Imprimer objet** permettent d'imprimer à chaque fois un nouvel ensemble de lignes. La commande **LISTBOX LIRE INFORMATION IMPRESSION** permet de contrôler le statut de l'impression durant l'impression.

Gestion des styles et des couleurs

Vous disposez de plusieurs possibilités pour définir des couleurs de fond, des couleurs de police et des styles de police dans les list box :

- au niveau des propriétés de l'objet list box,
- au niveau des propriétés de la colonne,
- en utilisant des tableaux ou des méthodes pour la list box et/ou pour chaque colonne,
- au niveau du texte de chaque cellule (si texte multistyle).

Des principes de priorité et d'héritage sont observés.

Priorité

Lorsqu'une même propriété de style ou de couleur est définie à plusieurs niveaux, l'ordre de priorité suivant est appliqué :

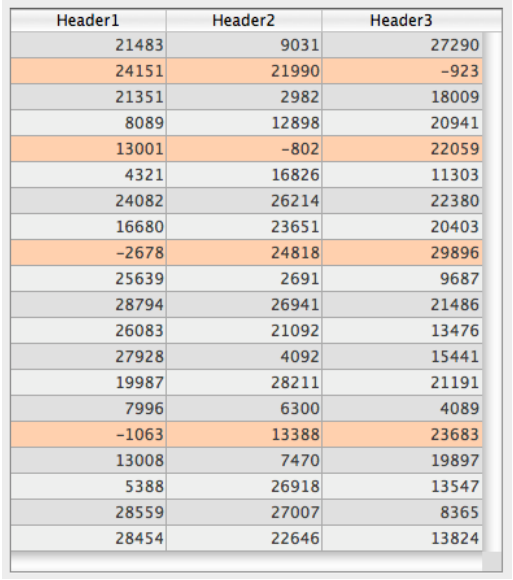
priorité élevée	Cellule (si texte multistyle)
	Tableaux/Méthodes colonne
	Tableaux/Méthodes list box
	Propriétés de colonne
priorité basse	Propriétés de list box

Par exemple, si vous définissez un style de caractères dans les propriétés de la list box et un autre via un tableau de styles pour la colonne, ce dernier sera

pris en compte.

Soit une list box dont les lignes ont une couleur alternée gris/gris clair, définies dans les propriétés de la list box. Un tableau de couleur de fond a également été défini pour la list box afin de passer en orange clair les lignes dont au moins une valeur est négative :

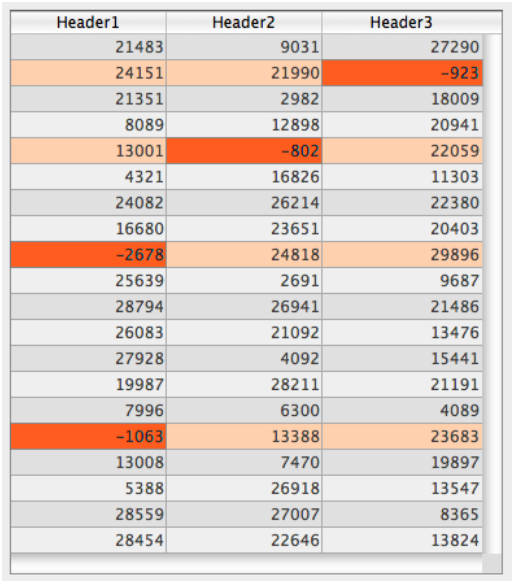
```
<>_BgndColors{$i}:=0x00FFD0B0    // orange
<>_BgndColors{$i}:=lk hérité  // valeur défaut
```



Header1	Header2	Header3
21483	9031	27290
24151	21990	-923
21351	2982	18009
8089	12898	20941
13001	-802	22059
4321	16826	11303
24082	26214	22380
16680	23651	20403
-2678	24818	29896
25639	2691	9687
28794	26941	21486
26083	21092	13476
27928	4092	15441
19987	28211	21191
7996	6300	4089
-1063	13388	23683
13008	7470	19897
5388	26918	13547
28559	27007	8365
28454	22646	13824

Vous souhaitez désormais afficher en fond orange foncé les cellule ayant une valeur négative. Pour cela, vous définissez un tableau de couleur de fond pour chaque colonne, par exemple *<>_BgndColor_1*, *<>_BgndColor_2* et *<>_BgndColor_3*. Les valeurs de ces tableaux seront prioritaires sur celles définies dans les propriétés de la list box et celles du tableau global de couleur de fond :

```
<>_BgndColorsCol_3{2}:=0x00FF8000    // orange foncé
<>_BgndColorsCol_2{5}:=0x00FF8000
<>_BgndColorsCol_1{9}:=0x00FF8000
<>_BgndColorsCol_1{16}:=0x00FF8000
```



Header1	Header2	Header3
21483	9031	27290
24151	21990	-923
21351	2982	18009
8089	12898	20941
13001	-802	22059
4321	16826	11303
24082	26214	22380
16680	23651	20403
-2678	24818	29896
25639	2691	9687
28794	26941	21486
26083	21092	13476
27928	4092	15441
19987	28211	21191
7996	6300	4089
-1063	13388	23683
13008	7470	19897
5388	26918	13547
28559	27007	8365
28454	22646	13824

Vous pouvez obtenir le même résultat à l'aide des commandes **LISTBOX FIXER STYLE LIGNE** et **LISTBOX FIXER COULEUR LIGNE**. Elles ont pour avantage d'éviter de devoir prédéfinir les tableaux de style/couleur des colonnes : ils sont créés dynamiquement par les commandes.

Héritage

Pour chaque attribut (style, couleur et couleur de fond), un héritage est mis en oeuvre lorsque la valeur par défaut est utilisée :

- pour les attributs des cellules : valeurs d'attributs des lignes
- pour les attributs des lignes : valeurs d'attributs des colonnes
- pour les attributs des colonnes : valeurs d'attributs de la list box

Ainsi, si vous souhaitez qu'un objet hérite de la valeur d'attribut du niveau supérieur, il vous suffit de passer la constante lk hérité (valeur par défaut) à la commande de définition ou directement dans l'élément de tableau de style/couleur correspondant.

Soit une list box contenant un style de caractère standard et des couleurs alternées :

standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard

Vous effectuez les modifications suivantes :

- le fond de la ligne 2 est passé en rouge via la propriété **Tableau couleurs de fond** de l'objet list box,
- le style de la ligne 4 est passé en italique via la propriété **Tableau de styles** de l'objet list box
- deux élements de la colonne 5 sont passés en gras via la propriété **Tableau de styles** de l'objet colonne 5
- les éléments 2 de la colonne 1 et 2 sont passés en fond bleu via la propriété **Tableau couleurs de fond** des objets colonne 1 et 2 :

standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	**standard**	standard
standard	*standard*	*standard*	*standard*	**standard**	*standard*
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard

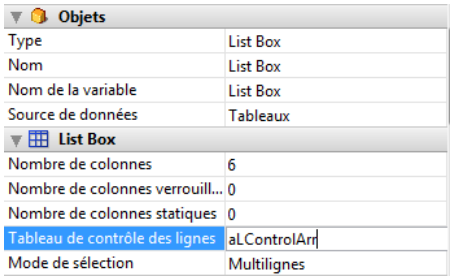
Pour restaurer l'apparence initiale de la list box, il suffit de :

- passer la constante lk hérité dans les éléments 2 des tableaux de fond des colonnes 1 et 2 : ils héritont alors de la couleur de fond rouge de la ligne.
- passer la constante lk hérité dans les éléments 3 et 4 des tableaux de style de la colonne 5 : ils héritont alors du style standard, hormis l'élément 4, qui passera en italique comme défini dans le tableau de style de la list box.
- passer la constante lk hérité dans l'élément 4 du tableau de style de la list box afin de supprimer le style italique.
- passer la constante lk hérité dans l'élément 2 du tableau de couleurs de fond de la list box afin de restaurer la couleur alternée d'origine de la list box.

Gestion de l'affichage des lignes

Des propriétés d'interface "masquée", "désactivée" et "sélectionnable" sont configurables pour chaque ligne de list box de type tableau.

Ces fonctionnalités sont gérées via le tableau de propriétés **Tableau de contrôle des lignes**, que vous pouvez désigner à l'aide de la commande **LISTBOX FIXER TABLEAU** ou via la Liste des propriétés :



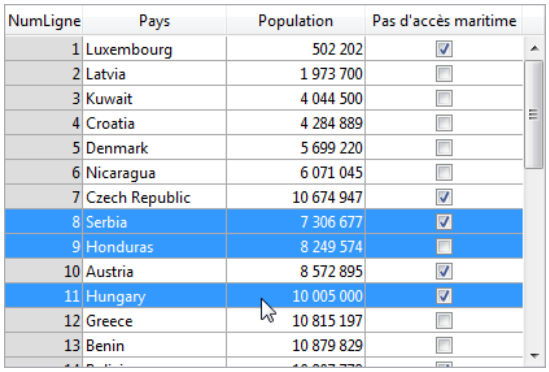
Le tableau de contrôle des lignes doit être de type Entier long et comporter le même nombre de lignes que la list box. Pour plus d'informations, reportez-vous à la section **Propriétés spécifiques des List box**.

Chaque élément du **Tableau de contrôle des lignes** définit le statut d'interface de la ligne correspondante dans la list box. Trois propriétés d'interface sont accessibles via les constantes du thème "List Box" :

Constante	Type	Valeur	Comment
lk ligne désactivée	Entier long	2	La ligne correspondante est désactivée. Les textes et les contrôles tels que les cases à cocher sont grisés ou estompés. Les zones de texte ne sont plus saisissables. Par défaut : activée
lk ligne masquée	Entier long	1	La ligne correspondante est masquée. Masquer des lignes affecte uniquement l'affichage de la list box. Les lignes masquées sont toujours présentes dans les tableaux et peuvent être manipulées par programmation. Les commandes du langage, notamment **LISTBOX Lire nombre lignes** ou **LISTBOX LIRE POSITION CELLULE**, ne tiennent pas compte de l'état masqué/affiché des lignes. Par exemple, dans une list box contenant 10 lignes et dont les 9 premières sont masquées, **LISTBOX Lire nombre lignes** retournera 10. Du point de vue de l'utilisateur, la présence de lignes masquées dans une list box n'est pas décelable visuellement. Seules les lignes visibles sont sélectionnables (par exemple via la commande Tout sélectionner). Par défaut : visible
lk ligne non sélectionnable	Entier long	4	La ligne correspondante n'est pas sélectionnable (le surlignage n'est plus possible). Les zones de texte ne sont plus saisissables à moins que l'option "Saisie sur clic unique" soit active. Les contrôles tels que les cases à cocher et les pop ups restent toutefois fonctionnels. Ce paramétrage est ignoré si le mode de sélection de la list box est "Aucun". Par défaut : sélectionnable

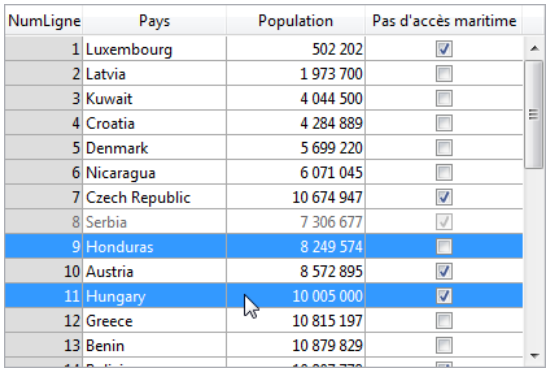
Pour modifier le statut d'une ligne, il vous suffit de passer la ou les constante(s) appropriée(s) dans l'élément correspondant. Par exemple, si vous souhaitez que la ligne n°10 ne soit pas sélectionnable, vous pouvez écrire :

```
aLControlArr{10}:=lk ligne non sélectionnable
```



Vous pouvez définir plusieurs propriétés d'interface en un seul appel :

```
aLControlArr{8}:=lk ligne non sélectionnable+lk ligne désactivée
```



NumLigne	Pays	Population	Pas d'accès maritime
1	Luxembourg	502 202	☑
2	Latvia	1 973 700	☐
3	Kuwait	4 044 500	☐
4	Croatia	4 284 889	☐
5	Denmark	5 699 220	☐
6	Nicaragua	6 071 045	☐
7	Czech Republic	10 674 947	☑
8	Serbia	7 306 677	☑
9	Honduras	8 249 574	☐
10	Austria	8 572 895	☑
11	Hungary	10 005 000	☑
12	Greece	10 815 197	☐
13	Benin	10 879 829	☐

A noter que la définition d'une propriété pour un élément écrase toutes les autres valeurs définies pour cet élément (si elles n'ont pas été réinitialisées). Par exemple :

```
aLControlArr{6}:=lk ligne désactivée+lk ligne non sélectionnable //la ligne 6 est désactivée et non sélectionnable
aLControlArr{6}:=lk ligne désactivée //la ligne 6 est toujours désactivée mais redevient sélectionnable
```

Affichage du résultat d'une requête SQL dans une List box

Il est possible de placer directement le résultat d'une requête SQL dans une list box de type tableau. Cette fonction offre un moyen rapide de visualiser le résultat des requêtes SQL. Seules les requêtes de type **SELECT** peuvent être utilisées. Ce mécanisme n'est pas utilisable avec une base SQL externe.

Les principes de mise en oeuvre sont les suivants :

- Vous créez la list box devant recevoir le résultat de la requête. La source de données de la list box doit être **Tableaux**.
- Vous exécutez la requête SQL de type **SELECT** et assignez le résultat à la variable associée la list box. Vous pouvez utiliser les mots-clés **Debut SQL**/**Fin SQL** (cf. manuel *Langage* de 4D).
- Les colonnes de la list box sont triables et modifiables par l'utilisateur.
- Chaque nouvelle exécution d'une requête **SELECT** avec la list box provoque la réinitialisation des colonnes (il n'est pas possible de remplir progressivement une même list box à l'aide de plusieurs requêtes **SELECT**).
- Il est préférable de placer dans la list box autant de colonnes qu'il y aura de colonnes SQL dans le résultat de la requête SQL. Si le nombre de colonnes de la list box est inférieur à celui requis par la requête **SELECT**, des colonnes sont automatiquement ajoutées. Si le nombre de colonnes de la list box est supérieur à celui requis par la requête **SELECT**, les colonnes superflues sont automatiquement masquées.
  Note : Les colonnes ajoutées automatiquement sont liées à des **Variables dynamiques** de type tableau. La durée de vie de ces tableaux dynamiques est celle du formulaire. Une variable dynamique est également créée pour chaque en-tête. Lorsque la commande **LISTBOX LIRE TABLEAUX** est appelée, le paramètre *tabVarCols* contient des pointeurs vers les tableaux dynamiques et le paramètre *tabVarEntêtes* contient des pointeurs vers les variables d'entête dynamiques. Si une colonne ajoutée est par exemple la cinquième, son nom est *sql_column5* et son nom d'en-tête *sql_header5*.
- En mode interprété, les tableaux existants et utilisés pourront être retypés automatiquement en fonction des données renvoyées par la requête SQL.

Exemple

Nous voulons récupérer tous les champs de la table PERSONS et placer leur contenu dans la list box dont le nom de variable est *vlistbox*. Dans la méthode objet d'un bouton (par exemple), il suffit d'écrire :

```
Debut SQL
   SELECT * FROM PERSONS INTO <<vlistbox>>
Fin SQL
```

Gestion des List box hiérarchiques

4D vous permet de définir et d'utiliser des list box hiérarchiques. Une list box hiérarchique est une list box dans laquelle le contenu de la première colonne apparaît sous forme hiérarchique. Ce type de représentation est adapté à la présentation d'informations comportant des valeurs répétées et/ou hiérarchiquement dépendantes (pays/région/ville...).

Seules les **list box de type tableau** peuvent être hiérarchiques.

Les list box hiérarchiques constituent un mode de représentation particulier des données, mais ne modifient pas la structure de ces données (les tableaux). Les list box hiérarchiques sont remplies et gérées exactement de la même manière que les list box non hiérarchiques (cf. **Gestion programmée des objets de type List box**).

Pour définir une list box hiérarchique, vous disposez des possibilités suivantes :

- configurer manuellement les éléments hiérarchiques via la Liste des propriétés ou à l'aide du pop up menu de gestion des list box, dans l'éditeur de formulaires. Ces points sont traités dans le manuel *Mode Développement* de 4D.
- utiliser les commandes **LISTBOX FIXER HIERARCHIE** et **LISTBOX LIRE HIERARCHIE**.

A la première ouverture d'un formulaire contenant une list box hiérarchique, par défaut toutes les lignes sont déployées.
Une ligne de rupture et un "noeud" hiérarchique sont automatiquement ajoutés dans la list box lorsque des valeurs sont répétées dans les tableaux. Par exemple, imaginons une list box contenant quatre tableaux définissant des villes, chaque ville étant caractérisée par un pays, une région, un nom et un nombre d'habitants :

Pays	Région	Ville	Population
France	Bretagne	Rennes	200000
France	Bretagne	Quimper	80000
France	Bretagne	Brest	120000
France	Normandie	Caen	75000
France	Normandie	Deauville	35000

Si cette list box est affichée sous forme hiérarchique (les trois premiers tableaux étant inclus dans la hiérarchie), vous obtenez :

Ville	Population
⊟ France	
⊟ Bretagne	
Rennes	200000
Quimper	80000
Brest	120000
⊟ Normandie	
Caen	75000
Deauville	35000

Les tableaux ne sont pas triés avant la construction de la hiérarchie. Si par exemple un tableau contient les données AAABBAACC, la hiérarchie obtenue sera :

> A
> B
> A
> C

Pour déployer ou contracter un "noeud" hiérarchique, cliquez dessus. Si vous effectuez **Alt+clic** (Windows) ou **Option+clic** (Mac OS) sur le noeud, tous ses sous-éléments seront déployés ou contractés. Ces opérations peuvent également être effectuées par programmation à l'aide des commandes **LISTBOX DEPLOYER** et **LISTBOX CONTRACTER**.

Gestion des sélections et des positions

Une list box hiérarchique affiche un nombre variable de lignes à l'écran en fonction de l'état déployé/contracté des noeuds hiérachiques. Cela ne signifie pas pour autant que le nombre de lignes des tableaux varie. Seul l'affichage est modifié, pas les données.

Il est important de comprendre ce principe car la gestion programmée des list box hiérarchiques se base toujours sur les données des tableaux, pas sur les données affichées. En particulier, les lignes de rupture ajoutées automatiquement ne sont pas prises en compte dans les tableaux d'options d'affichage (cf. ci-dessous le paragraphe).

Examinons par exemple les tableaux suivants :

France	Bretagne	Brest
France	Bretagne	Quimper
France	Bretagne	Rennes

Si ces tableaux sont représentés hiérarchiquement, la ligne "Quimper" ne sera pas affichée sur la deuxième ligne mais sur la quatrième, à cause des deux lignes de rupture ajoutées :

France
Bretagne
Brest
Quimper
Rennes

Quelle que soit la manière dont les données sont affichées dans la list box (hiérarchique ou non-hiérarchique), si vous souhaitez passer la ligne contenant "Quimper" en gras, vous devrez utiliser l'instruction TabStyle{2} = Gras. Seule la position de la ligne dans les tableaux est prise en compte.

Ce principe est mis en oeuvre pour les tableaux internes permettant de gérer :

- les couleurs
- les couleurs de fond
- les styles
- les lignes masquées
- les sélections

Par exemple, si vous voulez sélectionner la ligne contenant Rennes, vous devez passer :

```
->MaListbox{3}:=Vrai
```

Représentation non hiérarchique :

France	Bretagne	Brest
France	Bretagne	Quimper
France	Bretagne	Rennes

Représentation hiérarchique :

France
Bretagne
Brest
Quimper
Rennes

Note : Si une ou plusieurs lignes sont masquées du fait que leurs parents ont été contractés, elles ne sont plus sélectionnées. Seules les lignes visibles (directement ou suite à un défilement) sont sélectionnables. Autrement dit, les lignes ne peuvent pas être à la fois sélectionnées et cachées.

Tout comme pour les sélections, la commande **LISTBOX LIRE POSITION CELLULE** retournera les mêmes valeurs pour une list box hiérarchique et une list box non hiérarchique. Cela signifie que dans les deux exemples ci-dessous, **LISTBOX LIRE POSITION CELLULE** retournera la même position : (3;2)

Représentation non hiérarchique :

France	Bretagne	Brest	120000
France	Bretagne	Quimper	80000
France	Bretagne	Rennes	200000
France	Normandie	Caen	75000

Représentation hiérarchique :

France	
Bretagne	
Brest	120000
Quimper	80000
Rennes	200000
Normandie	
Caen	75000

Gestion des lignes de rupture

Si l'utilisateur sélectionne une ligne de rupture, **LISTBOX LIRE POSITION CELLULE** retourne la première occurrence de la ligne dans le tableau correspondant. Dans le cas suivant :

France	
Bretagne	
Brest	120000
Quimper	80000
Rennes	200000
Normandie	
Caen	75000

... **LISTBOX LIRE POSITION CELLULE** retourne (2;4). Pour sélectionner une ligne de rupture par programmation, vous devez utiliser la commande **LISTBOX SELECTIONNER RUPTURE**.

Les lignes de rupture ne sont pas prises en compte dans les tableaux internes permettant de gérer l'apparence graphique des list box (styles et couleurs). Il est toutefois possible de modifier ces caractéristiques pour les lignes de rupture via les commandes de gestion graphique des objets (thème **Objets (Formulaires)**). Il suffit pour cela d'exécuter ces commandes appropriées sur les tableaux constituant la hiérarchie.

Soit par exemple la list box suivante (les noms des tableaux associés sont précisés entre parenthèses) :

Représentation non hiérarchique :

(T1)	(T2)	(T3)	(T4)	(tStyle)	(tCouleur)
France	Bretagne	Brest	120000	*Normal*	*0*
France	Bretagne	Quimper	80000	*Souligné*	*0*
France	Bretagne	Rennes	200000	*Normal*	*0xFF0000*
France	Normandie	Caen	220000	*Normal*	*0*
France	Normandie	Deauville	4000	*Normal*	*0*

Représentation hiérarchique :

France	
Bretagne	
Brest	120000
Quimper	80000
Rennes	200000
Normandie	
Caen	75000
Deauville	4000

En mode hiérarchique, les niveaux de rupture ne sont pas pris en compte par les tableaux de modification de style nommés *tStyle* et *tCouleurs*. Pour modifier la couleur ou le style des niveaux de rupture, vous devez exécuter les instructions suivantes :

```
OBJET FIXER COULEURS RVB(T1;0x0000FF;0xB0B0B0)
OBJET FIXER STYLE POLICE(T2;Gras)
```

Note : Dans ce contexte, seule la syntaxe utilisant la variable tableau peut fonctionner avec les commandes de propriété d'objet car les tableaux n'ont alors pas d'objet associé.

Résultat :

France	
Bretagne	
Brest	120000
Quimper	80000
Rennes	200000
Normandie	
Caen	75000
Deauville	4000

Lignes masquées

Lorsque toutes les lignes d'une sous-hiérarchie sont masquées, la ligne de rupture est automatiquement masquée. Dans l'exemple ci-dessus, si les lignes 1 à 3 sont masquées, la ligne de rupture "Bretagne" n'apparaîtra pas.

Gestion optimisée du déployer/contracter

Vous pouvez optimiser l'affichage et la gestion des list box hiérarchiques en tirant parti des événements formulaire Sur déployer et Sur contracter.

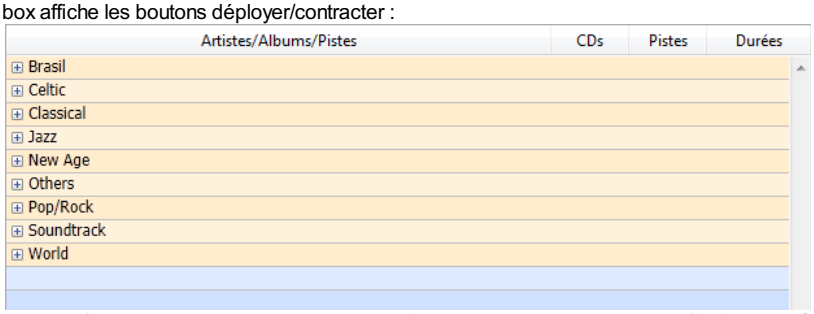
Une list box hiérarchique est construite à partir du contenu des tableaux qui la constituent, elle ne peut donc être affichée que lorsque tous les tableaux sont chargés en mémoire. Ce principe peut rendre difficile la génération de list box hiérarchiques de grande taille basées sur des tableaux générés à partir des données (via la commande **SELECTION VERS TABLEAU**), pour des raisons de rapidité d'affichage et d'utilisation de la mémoire.

L'emploi des événements formulaire Sur déployer et Sur contracter permet de s'affranchir de ces contraintes : il est possible de n'afficher qu'une partie de la hiérarchie et d'effectuer le chargement et le déchargement des tableaux à la volée, en fonction des actions de l'utilisateur.

Dans le contexte de ces événements, la commande **LISTBOX LIRE POSITION CELLULE** retourne la cellule sur laquelle l'utilisateur a cliqué afin de déployer ou de contracter une ligne.

Dans ce cas, le remplissage et le vidage des tableaux doivent être effectués par le code. Les principes à mettre en oeuvre sont :

- A l'affichage de la listbox, seul le premier tableau doit être rempli. Vous devez toutefois créer un second tableau avec des valeurs vides afin que la list box affiche les boutons déployer/contracter :



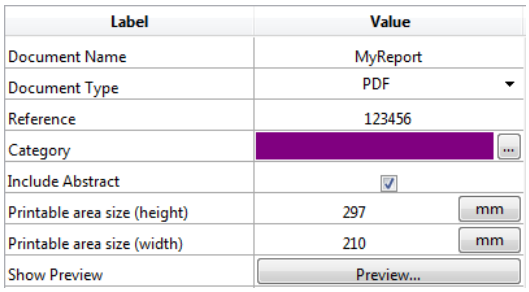
- Lorsque l'utilisateur clique sur un bouton de déploiement, vous pouvez traiter l'événement Sur déployer. La commande **LISTBOX LIRE POSITION CELLULE** retourne la cellule concernée et vous permet de construire la hiérarchie adéquate : vous alimentez le premier tableau avec des valeurs répétées et le second avec les valeurs issues de la commande **SELECTION VERS TABLEAU**, et vous insérez dans la list box autant de lignes que nécessaire à l'aide de la commande **LISTBOX INSERER LIGNES**.

- Lorsque l'utilisateur clique sur un bouton de contraction, vous pouvez traiter l'événement Sur contracter. La commande **LISTBOX LIRE POSITION CELLULE** retourne la cellule concernée : vous supprimez de la list box autant de lignes que nécessaire à l'aide de la commande **LISTBOX SUPPRIMER LIGNES**.

Utiliser des tableaux objets dans les colonnes (4D View Pro)

Les colonnes de list box peuvent être associées à des tableaux d'objets. Comme les tableaux d'objets peuvent contenir des données de types différents, cette puissante fonctionnalité vous permet de saisir et d'afficher divers types de valeurs dans les lignes d'une même colonne, ainsi que d'utiliser divers objets d'interface (*widgets*). Par exemple, vous pouvez placer une zone de saisie de texte dans la première ligne, une case à cocher dans la seconde, et une liste déroulante dans la troisième. Les tableaux d'objets vous donnent également accès à des widgets supplémentaires, tels que des boutons ou des sélecteurs de couleurs (*color picker*).

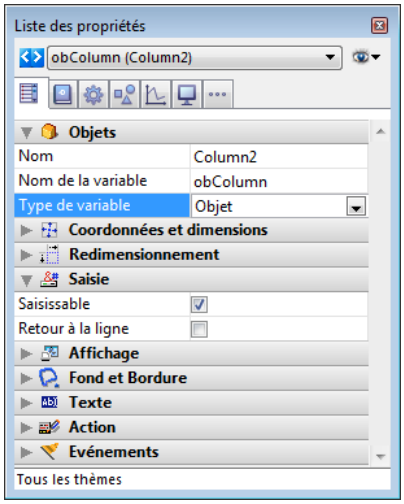
La list box suivante a été définie à l'aide d'un tableau d'objets :



A propos de 4D View Pro : La possibilité d'associer des tableaux d'objets aux colonnes de list box est une fonctionnalité "4D View Pro", ce qui signifie que son utilisation requiert une licence 4D View valide. 4D View Pro, nouvel outil en cours de développement, regroupe un ensemble de fonctionnalités relatives aux tableaux et aux présentations en listes. Il est destiné à remplacer progressivement le plug-in 4D View. Pour plus d'informations, veuillez vous reporter au site Web de 4D.

Configurer une colonne tableau d'objets

Pour affecter un tableau d'objets à une colonne de list box (list box de type tableau uniquement), il vous suffit de fournir le nom du tableau d'objets soit dans la Liste des propriétés (champ "Nom de variable"), soit à la commande **LISTBOX INSERER COLONNE**, comme pour toute colonne associée à un tableau. Dans la Liste des propriétés, vous pouvez sélectionner **Objet** comme "Type de variable" pour la colonne :



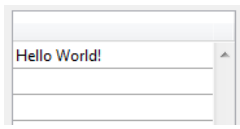
Les propriétés standard liées aux coordonnées, taille et style sont disponibles pour les colonnes de type objet. Elles peuvent être gérées à l'aide de la Liste des propriétés, ou en programmant les attributs de style, visibilité, couleur de police et de fond de chaque ligne de colonne objet de la list box. Ce type de colonne peut également être masqué.

Toutefois, le thème **Source de données** n'est pas disponible pour les colonnes objet des list box. En fait, le contenu de chaque cellule de la colonne est basé sur les attributs présents dans l'élément correspondant du tableau d'objets. Chaque élément du tableau peut définir :

- le type de valeur (*obligatoire*) : texte, couleur, événement, etc.
- la valeur elle-même (*optionnel*) : utilisé aussi bien pour la saisie que pour l'affichage
- le mode d'affichage du contenu de la cellule (*optionnel*) : bouton, liste, etc.
- des paramètres supplémentaires (*optionnel*) : dépend du type de valeur

Pour définir ces propriétés, vous devez placer les attributs adéquats dans l'objet (la liste des attributs disponibles est fournie ci-dessous). Par exemple, vous pouvez écrire "Hello World!" dans une colonne objet à l'aide de ce simple code :

```
TABLEAU OBJET(obColumn;0) //tableau de la colonne
C_OBJET($ob) //premier élément
OB FIXER($ob;"valueType";"text") //définit le type de valeur (obligatoire)
OB FIXER($ob;"value";"Hello World!") //définit la valeur
AJOUTER A TABLEAU(obColumn;$ob)
```



Note : Il n'est pas possible de choisir un format d'affichage et/ou un filtre de saisie pour les colonnes objet. Ces paramètres sont automatiquement définis en fonction du type de valeur.

valueType et affichage des données

Lorsqu'une colonne de list box est associée à un tableau d'objets, l'affichage, la saisie et l'édition des cellules sont basées sur l'attribut **valueType** présent dans chaque élément du tableau. Les valeurs **valueType** prises en charge sont les suivantes :

- "text" : pour une valeur texte
- "real" : pour une valeur numérique incluant des séparateurs tels que <espace>, <.> ou <,>
- "integer" : pour une valeur entière
- "boolean" : pour une valeur True/False
- "color" : pour définir une couleur de fond
- "event" : pour afficher un bouton avec un libellé

4D utilise des widgets par défaut en fonction de la valeur de "valueType" (par exemple, un "text" est affiché sous forme de zone de saisie de texte, un "boolean" est affiché sous forme de case à cocher, etc.), mais des représentations alternatives sont également disponibles via des options (par exemple, un "real" peut être affiché sous forme de menu déroulant). Le tableau suivant indique l'affichage par défaut ainsi que les variations possibles pour chaque type de valeur :

valueType	Widget par défaut	Widget(s) alternatif(s)
text	zone de saisie de texte	menu déroulant (enumération obligatoire) ou combo box (enumération)
real	zone de saisie de texte contrôlée (nombre et séparateurs)	menu déroulant (enumération obligatoire) ou combo box (enumération)
integer	zone de saisie de texte contrôlée (nombre)	menu déroulant (enumération obligatoire) ou combo box (enumération) ou case à cocher trois états
boolean	case à cocher	menu déroulant (enumération obligatoire)
color	couleur de fond	texte
event	bouton avec libellé	
		Tous les widgets peuvent associer un **unit toggle button** ou **ellipsis button** à la cellule.

Vous définissez l'affichage de la cellule et les variations à l'aide d'attributs spécifiques dans chaque objet (voir ci-dessous).

Formats d'affichage et filtres de saisie

Il n'est pas possible de choisir un format d'affichage et/ou un filtre de saisie pour les colonnes objet des list box, ces paramètres sont automatiquement définis en fonction du type de valeur. Ils sont listés dans le tableau suivant :

valueType	Format défaut	Contrôle de saisie
text	le même que celui de l'objet	pas de contrôle (tout caractère accepté)
real	le même que celui de l'objet (utilisation du séparateur décimal système)	"0-9", "." et "-"
		"0-9" et "." si min>=0
integer	le même que celui de l'objet	"0-9" et "-"
		"0-9" si min>=0
Boolean	case à cocher	N/A
color	N/A	N/A
event	N/A	N/A

Attributs

Chaque élément du tableau d'objets est un objet qui peut contenir un ou plusieurs attributs qui définiront le contenu de la cellule et l'affichage des données (voir exemple ci-dessus).

L'unique attribut obligatoire est "valueType" et ses valeurs acceptées sont "text", "real", "integer", "boolean", "color" et "event". Le tableau suivant liste tous les attributs acceptés dans les tableaux d'objets des list box, suivant la valeur de "valueType" (tout autre attribut est ignoré). Les formats d'affichage et des exemples sont fournis ci-dessous.

	valueType	text	real	integer	boolean	color	event
Attributs	***Description***						
value	valeur de la cellule (saisie ou affichage)	x	x	x			
min	valeur minimum		x	x			
max	valeur maximum		x	x			
behavior	valeur "threeStates"			x			
requiredList	menu déroulant défini dans l'objet	x	x	x			
choiceList	combo box défini dans l'objet	x	x	x			
requiredListReference	RefList 4D, dépend de la valeur de "saveAs"	x	x	x			
requiredListName	nom d'énumération 4D, dépend de la valeur de "saveAs"	x	x	x			
saveAs	"reference" ou "value"	x	x	x			
choiceListReference	RefList 4D, affiche une combo box	x	x	x			
choiceListName	nom d'énumération 4D, affiche une combo box	x	x	x			
unitList	tableau de X éléments	x	x	x			
unitReference	indice de l'élément sélectionné	x	x	x			
unitsListReference	RefList 4D pour les unités	x	x	x			
unitsListName	nom d'énumération 4D pour les unités	x	x	x			
alternateButton	ajouter un bouton alternatif	x	x	x	x	x	

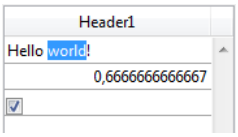
value

La valeur des cellules est stockée dans l'attribut "value". Cet attribut est utilisé pour la saisie (entrée) et pour l'affichage (sortie). Il peut également être utilisé

pour définir des valeurs par défaut lors de l'utilisation des listes (voir ci-dessous).

Exemple :

```
TABLEAU OBJET(obColumn;0) //tableau de la colonne
C_OBJET($ob1)
$entry:="Hello world!"
OB FIXER($ob1;"valueType";"text")
OB FIXER($ob1;"value";$entry) // si l'utilisateur saisit une valeur, $entry contiendra la nouvelle valeur
C_OBJET($ob2)
OB FIXER($ob2;"valueType";"real")
OB FIXER($ob2;"value";2/3)
C_OBJET($ob3)
OB FIXER($ob3;"valueType";"boolean")
OB FIXER($ob3;"value";Vrai)

AJOUTER A TABLEAU(obColumn;$ob1)
AJOUTER A TABLEAU(obColumn;$ob2)
AJOUTER A TABLEAU(obColumn;$ob3)
```



Note : La valeur Null est acceptée, elle définit une cellule vide.

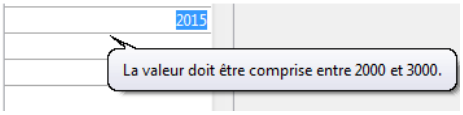
min et max

Lorsque "valueType" est "real" ou "integer", l'objet accepte également des attributs **min** et **max** avec des valeurs appropriées (les valeurs doivent être du même type que le "valueType").

Ces attributs peuvent être utilisés pour définir un intervalle de valeurs saisissables. Lorsqu'une cellule est validée (lorsqu'elle perd le focus), si la valeur saisie est inférieure à la valeur **min** ou supérieure à la valeur **max**, elle est alors rejetée. Dans ce cas, la valeur précédente est réappliquée et une info-bulle d'explication est affichée.

Exemple :

```
C_OBJET($ob3)
$entry3:=2015
OB FIXER($ob3;"valueType";"integer")
OB FIXER($ob3;"value";$entry3)
OB FIXER($ob3;"min";2000)
OB FIXER($ob3;"max";3000)
```



behavior

L'attribut **behavior** propose des variations de la représentation standard des valeurs. Dans la version actuelle de 4D, une seule variation est possible :

Attribut	Valeur(s) disponible(s)	valueType(s)	Description
behavior	threeStates	integer	Représente une valeur numérique sous forme de case à cocher à trois états. 2=intermédiaire, 1=cochée, 0=non cochée, -1=invisible, -2=non cochée désactivée, -3=cochée désactivée, -4=intermédiaire désactivée

Example :

```
C_OBJET($ob3)
OB FIXER($ob3;"valueType";"integer")
OB FIXER($ob3;"value";-3)
C_OBJET($ob4)
OB FIXER($ob4;"valueType";"integer")
OB FIXER($ob4;"value";-3)
OB FIXER($ob4;"behavior";"threeStates")
```

requiredList et choiceList

Lorsqu'un attribut "choiceList" ou "requiredList" est présent dans l'objet, la zone de saisie de texte est remplacée par une liste déroulante ou une combo box, en fonction de l'attribut :

- Si l'attribut est "choiceList", la cellule est affichée sous forme de combo box. Cela signifie que l'utilisateur peut sélectionner ou saisir une valeur.
- Si l'attribut est "requiredList", la cellule est affichée sous forme de liste déroulante. Cela signifie que l'utilisateur peut uniquement sélectionner une des valeurs de la liste.

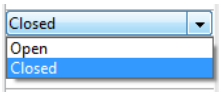
Dans les deux cas, vous pouvez utiliser un attribut "value" pour présélectionner une valeur dans le widget.

Note : Les valeurs du widget sont définies via un tableau. Si vous souhaitez associer le widget à une énumération 4D existante, vous devez utiliser les attributs "requiredListReference", "requiredListName", "choiceListReference" ou "choiceListName".

Exemples :

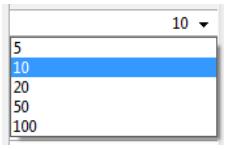
- Vous voulez afficher une liste déroulante avec juste deux options, "Open" ou "Closed". "Closed" doit être présélectionné :

```
TABLEAU TEXTE($RequiredList;0)
AJOUTER A TABLEAU($RequiredList;"Open")
AJOUTER A TABLEAU($RequiredList;"Closed")
C_OBJET($ob)
OB FIXER($ob;"valueType";"text")
OB FIXER($ob;"value";"Closed")
OB FIXER TABLEAU($ob;"requiredList";$RequiredList)
```



- Vous voulez accepter toute valeur entière, mais afficher une combo box contenant les valeurs les plus communes :

```
TABLEAU ENTIER LONG($ChoiceList;0)
AJOUTER A TABLEAU($ChoiceList;5)
AJOUTER A TABLEAU($ChoiceList;10)
AJOUTER A TABLEAU($ChoiceList;20)
AJOUTER A TABLEAU($ChoiceList;50)
AJOUTER A TABLEAU($ChoiceList;100)
C_OBJET($ob)
OB FIXER($ob;"valueType";"integer")
OB FIXER($ob;"value";10) //10 comme valeur par défaut
OB FIXER TABLEAU($ob;"choiceList";$ChoiceList)
```



requiredListName et requiredListReference

Les attributs "requiredListName" et "requiredListReference" vous permettent d'utiliser, dans une cellule de list box, une énumération définie dans 4D soit en mode Développement (via l'éditeur d'**Enumérations** de la Boîte à outils) soit par programmation (à l'aide de la commande **Nouvelle liste**). La cellule sera alors affichée sous forme de liste déroulante, ce qui signifie que l'utilisateur pourra uniquement choisir une des valeurs fournies dans la liste.

Utilisez "requiredListName" ou "requiredListReference" en fonction de la provenance de la liste : si la liste provient de la Boîte à outils, utilisez son nom ; sinon, si la liste a été définie par programmation, passez sa référence. Dans les deux cas, vous pouvez utiliser un attribut "value" pour présélectionner une valeur dans le widget.

Note: Si vous souhaitez définir des valeurs d'énumération via un simple tableau, vous pouvez utiliser l'attribut "requiredList".

Dans ce contexte, l'attribut "saveAs" peut être utilisé afin de définir si l'élément sélectionné doit être sauvegardé en tant que valeur ("value") ou en tant que référence ("reference") :

- Si "saveAs" = "reference", l'élément sera stocké en tant que référence ; "valueType" doit être de type "real" ou "integer".
- Si "saveAs" = "value", l'élément sera stocké en tant que valeur. Dans ce cas, "valueType" doit être du même type de les valeurs de la liste, généralement "text" ou "integer" ; sinon, 4D tentera de convertir la valeur de la liste en "valueType" de l'objet (voir exemples ci-dessous).

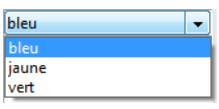
Pour plus d'informations sur l'option "save as", reportez-vous à la section **Enregistrer comme Valeur ou Référence** du manuel *Mode Développement*.

Note : Si la liste contient du texte représentant des valeurs réelles, le séparateur décimal doit être le point (".), quels que soient les paramètres locaux, ex : "17.6" "1234.456".

Exemples :

- Vous voulez afficher une liste déroulante basée sur une énumération nommée "colors" définie dans la Boîte à outils (contenant les valeurs "bleu", "jaune" et "vert"), la stocker en tant que valeur et afficher "bleu" par défaut :

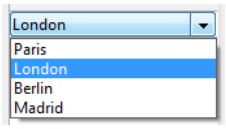

```
C_OBJET($ob)
OB FIXER($ob;"valueType";"text")
OB FIXER($ob;"saveAs";"value")
OB FIXER($ob;"value";"bleu")
OB FIXER($ob;"requiredListName";"colors")
```



- Vous voulez afficher une liste déroulante basée sur une liste créée par programmation, et la stocker en tant que référence :

```
<>List:=Nouvelle liste
AJOUTER A LISTE(<>List;"Paris";1)
AJOUTER A LISTE(<>List;"London";2)
```

```
AJOUTER A LISTE(<>List;"Berlin";3)
AJOUTER A LISTE(<>List;"Madrid";4)
C_OBJET($ob)
OB FIXER($ob;"valueType";"integer")
OB FIXER($ob;"saveAs";"reference")
OB FIXER($ob;"value";2) //affiche London by default
OB FIXER($ob;"requiredListReference";<>List)
```



choiceListName et choiceListReference

Les attributs "choiceListName" et "choiceListReference" permettent d'utiliser, dans une cellule de list box, une énumération définie dans 4D soit en mode Développement (via l'éditeur de la Boîte à outils) soit par programmation (à l'aide de la commande **Nouvelle liste**). La cellule sera alors affichée sous forme de combo box, ce qui signifie que l'utilisateur pourra choisir une des valeurs de la liste ou en saisir une.

Utilisez "choiceListName" ou "choiceListReference" en fonction de la provenance de la liste : si la liste provient de la Boîte à outils, utilisez son nom ; sinon, si la liste a été définie par programmation, passez sa référence. Dans les deux cas, vous pouvez utiliser un attribut "value" pour présélectionner une valeur dans le widget.

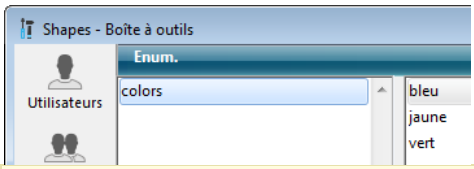
Note : Si vous souhaitez définir des valeurs d'énumération via un simple tableau, vous pouvez utiliser l'attribut "choiceList".

Dans ce contexte, l'attribut "saveAs" ne peut pas être utilisé car les éléments sélectionnés sont automatiquement stockés en tant que valeurs (cf.).

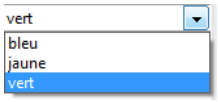
Note : Si la liste contient du texte représentant des valeurs réelles, le séparateur décimal doit être le point ("."), quels que soient les paramètres locaux, ex : "17.6" "1234.456".

Exemple :

Vous voulez afficher une combo box basée sur une énumération nommée "colors" définie dans la Boîte à outils (contenant les valeurs "bleu", "jaune" et "vert") et afficher "vert" par défaut :




```
C_OBJET($ob)
OB FIXER($ob;"valueType";"text")
OB FIXER($ob;"value";"vert")
OB FIXER($ob;"choiceListName";"colors")
```



unitsList, unitsListName, unitsListReference et unitReference

Vous pouvez utiliser des attributs spécifiques afin d'associer des unités aux valeurs des cellules (par exemple "10 cm", "20 pixels", etc.). Pour définir une liste d'unités, vous pouvez utiliser l'un des attributs suivants :

- "unitsList" : un tableau contenant les *x* éléments définissant les unités disponibles (ex : "cm", "pouces", "km", "miles", etc.). Utilisez cet attribut pour définir des unités dans l'objet.
- "unitsListReference" : une référence de liste 4D contenant les unités disponibles. Utilisez cet attribut pour définir des unités à l'aide d'une liste 4D créée avec la commande **Nouvelle liste**.
- "unitsListName" : un nom d'énumération 4D créée en mode Développement contenant les unités disponibles. Utilisez cet attribut pour définir des unités à l'aide d'une énumération 4D créée dans la Boîte à outils.

Quel que soit son mode de définition, la liste d'unités peut être associée à l'attribut suivant :

- "unitReference" : une valeur simple contenant l'indice (de 1 à *x*) de l'élément sélectionné dans la liste de valeurs "unitList", "unitsListReference" ou "unitsListName".

L'unité courante est affichée sous forme de bouton affichant successivement les valeurs de "unitList", "unitsListReference" ou "unitsListName" à chaque clic (par exemple "pixels" -> "lignes" -> "cm" -> "pixels" -> etc.)

Exemple : Vous souhaitez définir une valeur de saisie numérique suivie d'une unité parmi deux possibles : "cm" ou "pixels". La valeur courante est "2" + "cm". Vous utilisez des valeurs définies directement dans l'objet (attribut "unitsList") :

```
TABLEAU TEXTE($_units;0)
AJOUTER A TABLEAU($_units;"cm")
AJOUTER A TABLEAU($_units;"pixels")
C_OBJET($ob)
OB FIXER($ob;"valueType";"integer")
OB FIXER($ob;"value";2) // 2 "units"
OB FIXER($ob;"unitReference";1) //"cm"
OB FIXER TABLEAU($ob;"unitsList";$_units)
```

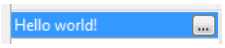
alternateButton

Si vous souhaitez ajouter un bouton d'ellipse [...] dans une cellule, il suffit de passer l'attribut "alternateButton" avec la valeur **vrai** dans l'objet. Le bouton sera automatiquement affiché dans la cellule.

Lorsque l'utilisateur clique sur ce bouton, un événement #cst id="2074097"/] est généré, vous permettant de traiter cette action comme vous le souhaitez (reportez-vous ci-dessous au paragraphe "**Gestion des événements**" pour plus d'informations).

Exemple :

```
C_OBJET($ob1)
$entry:="Hello world!"
OB FIXER($ob;"valueType";"text")
OB FIXER($ob;"alternateButton";Vrai)
OB FIXER($ob;"value";$entry)
```



valueType color

L'attribut "valueType" de valeur "color" vous permet d'afficher soit une couleur, soit un texte.

- Si la valeur est un nombre, un rectangle de couleur est dessiné à l'intérieur de la cellule. Exemple :

```
C_OBJET($ob4)
OB FIXER($ob4;"valueType";"color")
OB FIXER($ob4;"value";0x00FF0000)
```

- Si la valeur est un texte, le texte est simplement affiché (par exemple : "value";"Automatic").

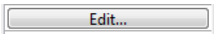
valueType event

L'attribut "valueType" de valeur "event" affiche un bouton qui génère simplement un événement Sur clic lorsque l'utilisateur clique dessus. Aucune donnée ou valeur ne peut être passée ou retournée.

Optionnellement, il est possible de passez un attribut "label" afin de donner un libellé au bouton.

Exemple :

```
C_OBJET($ob)
OB FIXER($ob;"valueType";"event")
OB FIXER($ob;"label";"Edit...")
```



Gestion des événements

Plusieurs événements peuvent être gérés dans les colonnes de list box de type tableau d'objets (cf. ci-dessus). Voici une synthèse des événements spécifiques :

- **Sur données modifiées** : L'événement Sur données modifiées est généré en cas de modification d'une valeur de la colonne, quel que soit le widget :
  - zone de saisie de texte
  - listes déroulante
  - zone de combo box
  - bouton d'unité (passage valeur x à valeur x+1)
  - case à cocher (passage cochée/non cochée)
- **Sur clic** : Lorsque l'utilisateur clique sur un bouton installé à l'aide de l'attribut "valueType" "event", un événement Sur clic est généré. Cet événement doit être ensuite géré par le programmeur.
- **Sur clic alternatif** : Lorsque l'utilisateur clique sur un bouton d'ellipse (attribut "alternateButton"), un événement Sur clic alternatif est généré. Cet événement doit être ensuite géré par le programmeur.

Note de compatibilité (4D v15) : Sur clic alternatif est le nouveau nom de l'événement Sur clic flèche, renommé afin de souligner l'extension de son champ d'action.

A propos de 4D View Pro

4D View Pro est un nouvel outil, développé par 4D. Essentiellement basé sur les list box, il fournit un ensemble de fonctionnalités avancées relatives aux tableaux et aux présentations de données en liste. 4D View Pro est destiné à proposer aux utilisateurs de 4D une alternative moderne et intégrée à certaines fonctionnalités du précédent produit 4D View.

Listes des fonctionnalités de 4D View Pro

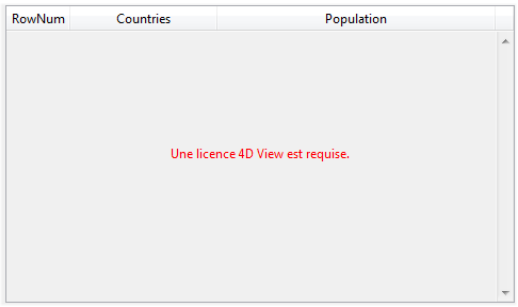
Dans la version actuelle du programme, les fonctionnalités 4D View Pro sont les suivantes :

- Tableaux objet associés aux colonnes de list box (cf. **Utiliser des tableaux objets dans les colonnes (4D View Pro)**)
- Hauteur de ligne variable dans une list box :
  - propriété **Tableau hauteurs de lignes**
  - commandes **LISTBOX Lire hauteur ligne** et **LISTBOX FIXER HAUTEUR LIGNE**
  - constante lk tableau hauteurs lignes pour les commandes **LISTBOX Lire tableau** et **LISTBOX FIXER TABLEAU**

Installation et activation

A la différence du précédent produit 4D View, les fonctionnalités de 4D View Pro sont directement intégrées dans l'application 4D elle-même, ce qui facilite le déploiement et la gestion des versions. Aucune installation additionnelle n'est requise.

Toutefois, 4D View Pro nécessite la même licence que 4D View. Cette licence doit être installée dans votre application pour que les fonctionnalités 4D View Pro soient actives. A l'exécution, si une list box utilisant une fonctionnalité 4D View Pro est affichée alors que la licence 4D View n'est pas présente, le contenu de la list box n'apparaît pas et un message d'erreur est affiché à sa place :



LISTBOX CONTRACTER

LISTBOX CONTRACTER ({* ;} objet {; récursive {; sélecteur {; ligne {; colonne}}}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Variable (si * omis)
récursive	Booléen	⇒	Vrai = contracter les sous-niveaux, Faux = ne pas contracter les sous-niveaux
sélecteur	Entier long	⇒	Partie de la list box à contracter
ligne	Entier long	⇒	Numéro de ligne de la rupture à contracter ou Numéro de niveau de la list box à contracter
colonne	Entier long	⇒	Numéro de colonne de la rupture à contracter

Description

La commande **LISTBOX CONTRACTER** vous permet de provoquer la contraction des lignes de rupture de l'objet list box désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Si la list box n'est pas configurée en mode hiérarchique, la commande ne fait rien. Pour plus d'informations sur les list box hérarchiques, reportez-vous à la section **Gestion des List box hiérarchiques**.

Le paramètre optionnel *récursive* vous permet de paramétrer la contraction des sous-niveaux hiérarchiques de la list box. Passez Vrai ou omettez ce paramètre pour que la commande provoque la contraction de tous les niveaux et tous les sous-niveaux. Si vous passez Faux, seul le premier niveau sera contracté.

Le paramètre optionnel *sélecteur* vous permet de définir la portée de la commande. Vous pouvez passer dans ce paramètre l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk tout	Entier long	0	La commande agit sur tous les sous-niveaux (valeur par défaut, utilisée si le paramètre est omis).
lk sélection	Entier long	1	La commande agit sur les sous-niveaux sélectionnés.
lk ligne rupture	Entier long	2	La commande agit sur le sous-niveau auquel appartient la "cellule" désignée par les paramètres *ligne* et *colonne*. A noter que ces paramètres représentent les numéros de ligne et de colonne dans la listbox en mode standard et non dans sa représentation hiérarchique. Si les paramètres *ligne* et *colonne* sont omis, la commande ne fait rien.
lk niveau	Entier long	3	La commande agit sur toutes les lignes de rupture correspondant à la colonne *niveau*. Ce paramètre désigne un numéro de colonne dans la listbox en mode standard et non dans sa représentation hiérarchique. Si le paramètre *niveau* est omis, la commande ne fait rien.

Si la sélection ou la list box ne contient pas de ligne de rupture, ou si toutes les lignes de rupture sont déjà contractées, la commande ne fait rien.

Exemple

Cet exemple contracte le premier niveau de lignes de rupture de la sélection de la list box :

```
LISTBOX CONTRACTER(*;"MaListbox";Faux;lk sélection)
```

LISTBOX DEPLACER COLONNE

LISTBOX DEPLACER COLONNE ({* ;} objet ; positionCol)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis) de la colonne à déplacer
positionCol	Entier long	⇒	Nouvel emplacement de la colonne

Description

La commande **LISTBOX DEPLACER COLONNE** permet de déplacer par programmation la colonne désignée par le(s) paramètre(s) *objet* et * dans le contexte du formulaire en cours d'exécution (mode Application). Le formulaire d'origine, généré en mode Développement, n'est pas modifié.

Les paramètres *objet* et * désignent la colonne à déplacer. Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom de colonne (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable de colonne. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

La colonne est déplacée juste avant la colonne désignée par le paramètre *positionCol*. Si le paramètre *positionCol* est supérieur au nombre total de colonnes, la colonne est déplacée après la dernière colonne.

Note : Cette commande ne fait rien si elle est appliquée à la première colonne d'une list box affichée en mode hiérarchique.

La commande tient compte des propriétés de colonnes statiques et verrouillées : si vous tentez par exemple de déplacer une colonne statique, la commande ne fait rien.

Cette fonctionnalité est présente dans 4D en mode Application : l'utilisateur peut déplacer des colonnes non statiques à l'aide de la souris. En revanche, à la différence du déplacement effectué par l'utilisateur, la commande ne génère pas l'événement Sur déplacement colonne.

Exemple

Vous souhaitez intervertir les 2e et 3e colonnes de la list box :

```
LISTBOX DEPLACER COLONNE(*;"colonne2";3)
```

LISTBOX DEPLOYER

LISTBOX DEPLOYER ({* ;} objet {; récursive {; sélecteur {; ligne {; colonne}}}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
récursive	Booléen	⇒	Vrai = déployer les sous-niveaux, Faux = ne pas déployer les sous-niveaux
sélecteur	Entier long	⇒	Partie de la list box à déployer
ligne	Entier long	⇒	Numéro de ligne de la rupture à déployer ou Numéro de niveau de la list box à déployer
colonne	Entier long	⇒	Numéro de colonne de la rupture à déployer

Description

La commande **LISTBOX DEPLOYER** vous permet de provoquer le déploiement des lignes de rupture de l'objet list box affiché en mode hiérarchique désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Si la list box n'est pas configurée en mode hiérarchique, la commande ne fait rien. Pour plus d'informations sur les list box hérarchiques, reportez-vous à la section **Gestion des List box hiérarchiques**.

Le paramètre optionnel *récursive* vous permet de paramétrer le déploiement des sous-niveaux hiérarchiques de la list box. Passez Vrai ou omettez ce paramètre pour que la commande provoque le déploiement de tous les niveaux et tous les sous-niveaux. Si vous passez Faux, seul le premier niveau désigné sera déployé.

Le paramètre optionnel *sélecteur* vous permet de définir la portée de la commande. Vous pouvez passer dans ce paramètre l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk tout	Entier long	0	La commande agit sur tous les sous-niveaux (valeur par défaut, utilisée si le paramètre est omis).
lk sélection	Entier long	1	La commande agit sur les sous-niveaux sélectionnés.
lk ligne rupture	Entier long	2	La commande agit sur le sous-niveau auquel appartient la "cellule" désignée par les paramètres *ligne* et *colonne*. A noter que ces paramètres représentent les numéros de ligne et de colonne dans la listbox en mode standard et non dans sa représentation hiérarchique. Si les paramètres *ligne* et *colonne* sont omis, la commande ne fait rien.
lk niveau	Entier long	3	La commande agit sur toutes les lignes de rupture correspondant à la colonne *niveau*. Ce paramètre désigne un numéro de colonne dans la listbox en mode standard et non dans sa représentation hiérarchique. Si le paramètre *niveau* est omis, la commande ne fait rien.

La commande ne sélectionne pas les lignes de rupture.

Si la sélection ou la list box ne contient pas de ligne de rupture, ou si toutes les lignes de rupture sont déjà déployées, la commande ne fait rien.

Exemple

Cet exemple illustre différents modes d'utilisation de la commande. Soient les tableaux suivants représentés dans une list box :

France	Brittany	Brest	120000
France	Brittany	Quimper	80000
France	Brittany	Rennes	200000
France	Normandy	Caen	220000
France	Normandy	Deauville	4000
France	Normandy	Cherbourg	41000
Belgium	Wallonia	Namur	111000
Belgium	Wallonia	Liege	200000
Belgium	Flanders	Antwerp	472000
Belgium	Flanders	Louvain	95000

```
 //Déployer toutes les lignes et sous-lignes de rupture de la list box
LISTBOX DEPLOYER(*;"MaListbox")
```

V France	
V Bretagne	
Brest	120000
Quimper	80000
Rennes	200000
V Normandie	
Caen	220000
Deauville	4000
Cherbourg	41000
V Belgique	
V Wallonie	
Namur	111000
Liège	200000
V Flandre	
Anvers	472000
Louvain	95000

```
  //Déployer le premier niveau de lignes de rupture de la sélection
LISTBOX DEPLOYER(*;"MaListbox";Faux;lk sélection)
  //Si la ligne "Belgique" était sélectionnée
```

> France
V Belgique
> Wallonie
> Flandre

```
  //Déployer la ligne de rupture Bretagne sans récursivité
LISTBOX DEPLOYER(*;"MaListbox";Faux;lk ligne rupture;1;2)
```

V France	
V Bretagne	
Brest	120000
Quimper	80000
Rennes	200000
> Normandie	
> Belgique	

```
  //Déployer toutes les premières colonnes (pays) sans récursivité
LISTBOX DEPLOYER(*;"MaListbox";Faux;lk niveau;1)
```

V France
> Bretagne
> Normandie
V Belgique
> Wallonie
> Flandre

LISTBOX DUPLIQUER COLONNE

LISTBOX DUPLIQUER COLONNE ({* ;} objet ; positionCol ; nomCol ; variableCol ; nomEntête ; varEntête {; nomPied ; variablePied})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis) de la colonne à dupliquer
positionCol	Entier long	⇒	Emplacement de la nouvelle colonne dupliquée
nomCol	Chaîne	⇒	Nom de la nouvelle colonne
variableCol	Tableau, Champ, Variable, Pointeur nil	⇒	Nom de la variable tableau de la colonne ou champ ou variable
nomEntête	Chaîne	⇒	Nom d'objet de l'en-tête de la colonne
varEntête	Variable entier, Pointeur nil	⇒	Variable d'en-tête de la colonne
nomPied	Chaîne	⇒	Nom d'objet du pied de la colonne
variablePied	Variable, Pointeur nil	⇒	Variable du pied de la colonne

Description

La commande **LISTBOX DUPLIQUER COLONNE** permet de dupliquer la colonne désignée par le(s) paramètre(s) *objet* et * dans le contexte du formulaire en cours d'exécution (mode Application). Le formulaire d'origine, généré en mode Développement, n'est pas modifié.

Note : Cette fonctionnalité est présente dans 4D, en mode Développement uniquement, via la commande **Dupliquer colonne** du menu contextuel de l'éditeur de formulaires.

Par défaut, toutes les options de style (taille, couleur, formats, etc.) définies pour la colonne source via la Liste des propriétés ou les commandes de gestion d'objet (**OBJET FIXER COULEUR**, etc.) sont appliquées à la copie. La méthode objet et le paramétrage des événements formulaire sont également dupliqués.
En revanche, la source de données (tableau ou sélection, selon le type de source défini pour la list box) ainsi que les tableaux de style et de couleurs ne sont pas dupliqués. Il vous appartient de les définir pour chaque nouvelle colonne après la duplication.

Les paramètres *objet* et * désignent la colonne à dupliquer. Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom de colonne (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable de colonne. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Note : Cette commande ne fait rien si elle est appliquée à la première colonne d'une list box affichée en mode hiérarchique.

La nouvelle colonne dupliquée est placée juste avant la colonne désignée par le paramètre *positionCol*. Si le paramètre *positionCol* est supérieur au nombre total de colonnes, la colonne est dupliquée après la dernière colonne.

Passez dans les paramètres *nomCol* et *variableCol* le nom d'objet et la variable de la nouvelle colonne dupliquée.

- Dans le cadre d'une list box de type tableau, le nom de la variable correspond au nom du tableau dont le contenu sera affiché dans la colonne. Vous pouvez passer un pointeur Nil (->[]) dans un contexte dynamique (cf. ci-dessous).
- Dans le cadre d'une list box de type sélection, vous pouvez passer un champ ou une variable dans le paramètre *variableCol*. Le contenu de la colonne sera alors la valeur du champ ou de la variable, évaluée pour chaque enregistrement de la sélection associée à la list box. Ce type de contenu ne peut être utilisé que lorsque la propriété "Source de données" de la list box est Sélection courante ou Sélection temporaire.

N'oubliez pas que la source de données de la colonne d'origine n'est pas dupliquée : le contenu de la variable, tableau ou champ source de la nouvelle colonne dupliquée doit être défini.

Passez dans les paramètres *nomEntête* et *variableEntête* le nom d'objet et la variable de l'en-tête de la nouvelle colonne dupliquée. Vous pouvez également passer dans les paramètres *nomPied* et *variablePied* le nom d'objet et la variable du pied de la colonne insérée. Si vous omettez le paramètre *variablePied*, 4D utilisera une variable dynamique.

Note : Les noms d'objets doivent être uniques dans un formulaire. Vous devez veiller à ce que les noms passés dans les paramètres *nomCol*, *nomEntête* et *nomPied* ne soient pas déjà utilisés. Sinon, la colonne n'est pas dupliquée et une erreur est générée.

Cette commande doit être utilisée dans le contexte de l'affichage d'un formulaire. Elle sera généralement appelée dans l'événement Sur chargement du formulaire ou suite à une action utilisateur (événement Sur clic).

Duplication dynamique

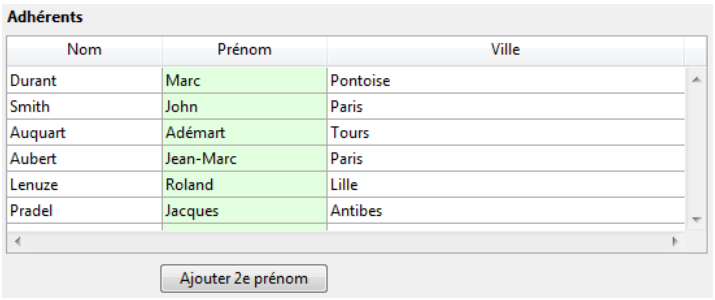
A compter de 4D v14 R3, vous pouvez dupliquer dynamiquement des colonnes de list box, 4D prenant automatiquement en charge les définitions de variables nécessaires (colonne, pied et en-tête).

Pour cela, **LISTBOX DUPLIQUER COLONNE** accepte un pointeur **Nil** (**->[]**) comme valeur pour les paramètres *variableCol* (list box de type tableau uniquement), *variableEntête* et *variablePied*. Dans ce cas, 4D va créer dynamiquement les variables requises lors de l'exécution de la commande (pour plus d'informations, reportez-vous à la section **Variables dynamiques**).

A noter que les variables d'en-tête et de pied sont toujours créées avec un type spécifique (respectivement entier long et texte). A l'inverse, les variables de colonne ne peuvent pas être typées à la création car les list box acceptent différents types de tableaux pour ces variables (tableau texte, tableau entier, etc.). Vous devez donc définir manuellement le type du tableau (cf. exemple 2). Il est important d'effectuer ce typage avant d'appeler des commandes telles que **LISTBOX INSERER LIGNES** pour insérer des nouveaux éléments dans le tableau. Ou bien, il est possible d'utiliser **AJOUTER A TABLEAU** pour à la fois typer le tableau et insérer des éléments.

Exemple 1

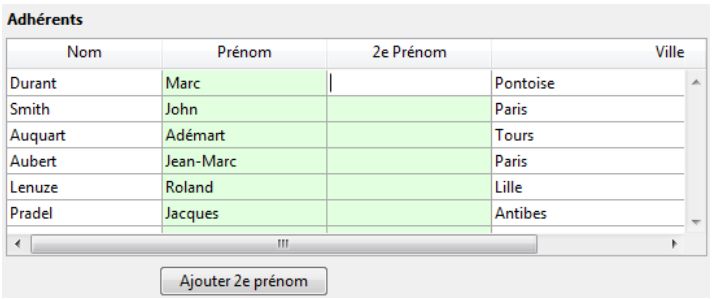
Dans une list box de type tableau, on souhaite dupliquer la colonne "Prénom", prête pour la saisie :



Le code du bouton :

```
TABLEAU TEXTE(tPrenoms2;Enregistrements dans table([Adhérents]))
LISTBOX DUPLIQUER COLONNE(*;"colonne2";3;"col2bis";tPrenoms2;"PrenomBis";vHead2Bis)
OBJET FIXER TITRE(*;"PrenomBis";"2e Prénom")
EDITER ELEMENT(*;"col2bis";0)
```

Lorsque vous cliquez sur le bouton, la list box apparaît ainsi :



Exemple 2

Vous souhaitez dupliquer dynamiquement une colonne booléenne et modifier son titre :

```
C_POINTEUR($ptr)
LISTBOX DUPLIQUER COLONNE(*;"colbool";3;"colboolDupl";$ptr;"HeaderboolDupli";$ptr;"footerboolDupli";$ptr)
colprt:=OBJET Lire pointeur(Objet nommé;"colboolDupl")
TABLEAU BOOLEEN(colprt->;10)
headprt:=OBJET Lire pointeur(Objet nommé;"HeaderboolDupli")
OBJET FIXER TITRE(headprt->;"Nouvelle colonne dupliquée")
```

LISTBOX FIXER CALCUL PIED

LISTBOX FIXER CALCUL PIED ({* ;} objet ; calcul)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
calcul	Entier long	⇒	Calcul pour la zone de pied

Description

La commande **LISTBOX FIXER CALCUL PIED** permet de définir le calcul automatique associé à la zone de pied de list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.
Le paramètre *objet* peut désigner :

- la variable ou le nom d'une zone de pied. Dans ce cas, la commande s'applique à cette zone.
- la variable ou le nom d'une colonne de list box. Dans ce cas, la commande s'applique à la zone de pied de cette colonne.
- la variable ou le nom d'une list box. Dans ce cas, la commande s'applique toutes les zones de pied de la list box.

Passez dans *calcul* une des constantes suivantes, placées dans le thème **List box pied calcul**, afin de définir le calcul à effectuer :

Constante	Type	Comment
lk pied écart type	Entier long	Utilisable avec les colonnes de type numérique, heure (listbox de type tableau uniquement) Type par défaut du résultat du calcul : Réel
lk pied max	Entier long	Utilisable avec les colonnes de type numérique, date, heure, booléen Type par défaut du résultat du calcul : type du tableau ou champ de la colonne
lk pied min	Entier long	Utilisable avec les colonnes de type numérique, date, heure, booléen Type par défaut du résultat du calcul : type du tableau ou champ de la colonne
lk pied moyenne	Entier long	Utilisable avec les colonnes de type numérique, heure Type par défaut du résultat du calcul : Réel
lk pied nombre	Entier long	Utilisable avec les colonnes de type numérique, texte, date, heure, booléen, image Type par défaut du résultat du calcul : Entier long
lk pied personnalisé	Entier long	Aucun calcul n'est effectué par 4D. La variable du pied doit être calculée par programmation. Type par défaut du résultat du calcul : type de la variable
lk pied somme	Entier long	Utilisable avec les colonnes de type numérique, heure, booléen Type par défaut du résultat du calcul : type du tableau ou champ de la colonne
lk pied somme des carrés	Entier long	Utilisable avec les colonnes de type numérique, heure (listbox de type tableau uniquement) Type par défaut du résultat du calcul : Réel
lk pied variance	Entier long	Utilisable avec les colonnes de type numérique, heure (listbox de type tableau uniquement) Type par défaut du résultat du calcul : Réel

A noter que les calculs prédéfinis tiennent compte de toutes les valeurs de la colonne, y compris les valeurs des lignes éventuellement masquées. Si vous souhaitez restreindre un calcul aux lignes visibles, vous devez utiliser la constante lk pied personnalisé et effectuer un calcul personnalisé.

Si le type de données de la colonne ou d'au moins une colonne de la list box (si *objet* désigne une list box) est incompatible avec le *calcul* défini, le pied n'est pas modifié et l'erreur 18 est générée. Si une colonne contient une formule (list box de type sélection), l'erreur 10 est générée.

Note : Les variables de zone de pied sont automatiquement typées (lorsqu'elles ne sont pas typées par programmation) en fonction du type de calcul défini dans la Liste des propriétés (cf. **Propriétés spécifiques des pieds de List box**). Si le type de la variable diffère du résultat attendu par la commande **LISTBOX FIXER CALCUL PIED**, une erreur de type est générée. Par exemple, pour une colonne affichant des dates, si le pied effectue un calcul 'Maximum', la variable *pied* sera typée en date. Si vous exécutez alors l'instruction **LISTBOX FIXER CALCUL PIED**(footer;lk pied nombre), une erreur est générée car le type du résultat attendu (entier long) est différent du type de la variable.

LISTBOX FIXER COLONNES STATIQUES

LISTBOX FIXER COLONNES STATIQUES ({* ;} objet ; nbColonnes)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
nbColonnes	Entier long	⇒	Nombre de colonnes à rendre statiques

Description

La commande **LISTBOX FIXER COLONNES STATIQUES** permet de rendre statiques les *nbColonnes* premières colonnes gauches de la list box désignée par les paramètres *objet* et *.

Les colonnes statiques (ou colonnes fixes) ne peuvent pas être déplacées dans la list box.

Note : Les colonnes statiques et les colonnes verrouillées sont deux fonctionnalités indépendantes. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

LISTBOX FIXER COLONNES VERROUILLEES

LISTBOX FIXER COLONNES VERROUILLEES ({* ;} objet ; nbColonnes)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
nbColonnes	Entier long	⇒	Nombre de colonnes à verrouiller

Description

La commande **LISTBOX FIXER COLONNES VERROUILLEES** permet de verrouiller les *nbColonnes* premières colonnes gauches de la list box désignée par les paramètres *objet* et *.

Les colonnes verrouillées restent affichées dans la partie gauche de la list box, elles ne défilent pas avec le reste de la list box. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Vous pouvez passer dans *nbColonnes* toute valeur comprise entre 1 et le nombre total de colonnes de la list box -1. Pour une list box ayant N colonnes, si vous passez dans *nbColonnes* une valeur > N-1, elle sera automatiquement réduite à la valeur N-1.

Si vous passez 0 ou une valeur négative dans *nbColonnes*, le verrouillage des colonnes est supprimé.

LISTBOX FIXER COULEUR GRILLE

LISTBOX FIXER COULEUR GRILLE ({* ;} objet ; couleur ; horizontal ; vertical)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
couleur	Entier long	⇒	Valeur de couleur RVB
horizontal	Booléen	⇒	Utiliser la couleur pour les traits horizontaux
vertical	Booléen	⇒	Utiliser la couleur pour les traits verticaux

Description

La commande **LISTBOX FIXER COULEUR GRILLE** permet de modifier la couleur de la grille de l'objet list box désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Passez dans le paramètre *couleur* une valeur de couleur RVB. Pour plus d'informations sur les couleurs RVB, reportez-vous à la description de la commande **OBJET FIXER COULEURS RVB**.

Les paramètres *horizontal* et *vertical* vous permettent de spécifier les traits auxquels la couleur doit être appliquée :

- si vous passez **Vrai** dans *horizontal*, la couleur sera appliquée aux traits horizontaux de la grille. Si vous passez **Faux**, leur couleur ne sera pas modifiée.
- si vous passez **Vrai** dans *vertical*, la couleur sera appliquée aux traits verticaux de la grille. Si vous passez **Faux**, leur couleur ne sera pas modifiée.

LISTBOX FIXER COULEUR LIGNE

LISTBOX FIXER COULEUR LIGNE ({* ;} objet ; ligne ; couleur {; typeCouleur})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ligne	Entier long	⇒	Numéro de ligne
couleur	Entier long	⇒	Valeur de couleur RVB
typeCouleur	Entier long	⇒	Listbox couleur de police (défaut) ou Listbox couleur de fond

Description

Note : Cette commande fonctionne uniquement avec les list box de type tableau.

La commande **LISTBOX FIXER COULEUR LIGNE** vous permet de définir une couleur pour une ligne ou une cellule de la list box tableau désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Vous pouvez désigner comme paramètre *objet* une list box ou une colonne de list box :

- si *objet* désigne une list box, la commande s'applique à la ligne
- si *objet* désigne une colonne, la commande s'applique à la cellule située à l'intersection colonne/ligne

Passez dans *ligne* le numéro de la ligne à laquelle la nouvelle couleur doit être appliquée.

Note : La commande ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.

Passez dans *couleur* une valeur de couleur RVB. Pour plus d'informations sur les couleurs RVB, reportez-vous à la description de la commande **OBJET FIXER COULEURS RVB**. Si vous souhaitez que la ligne hérite du paramétrage de couleur défini au niveau supérieur, passez la constante lk hérité dans *couleur*.

Passez la constante lk couleur de fond ou lk couleur de police dans le paramètre *typeCouleur* selon que vous souhaitez appliquer la couleur en tant que couleur de fond ou couleur de police de la ligne. Si vous omettez ce paramètre, la couleur est appliquée en tant que couleur de police.

Cette commande modifie les valeurs présentes dans les tableaux de couleurs éventuellement définis pour la colonne ou la listbox. Si ces tableaux ne sont pas déjà définis, la commande crée des tableaux dynamiques auxquels vous pourrez accéder à l'aide de la commande **LISTBOX Lire tableau**.

Si des valeurs de couleur contradictoires sont définies via les propriétés de la list box ou de la colonne, un ordre de priorité est appliqué. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Exemple

Dans une list box tableau, on souhaite définir des paramétrages de couleur pour une ligne et pour une cellule de cette ligne :

```
    // Définition de la couleur de police pour la cellule (jaune)
LISTBOX FIXER COULEUR LIGNE(*;"Col5";3;0x00FFFF00)

    // Définition de la couleur de fond et de police pour la ligne 3
    // fond rouge, police bleue
LISTBOX FIXER COULEUR LIGNE(*;"ListBox";3;0x00FF0000;lk couleur de fond)
LISTBOX FIXER COULEUR LIGNE(*;"List Box";3;0x000000FF)
```

text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

LISTBOX FIXER FORMULE COLONNE

LISTBOX FIXER FORMULE COLONNE ({* ;} objet ; formule ; typeDonnées)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
formule	Chaîne	⇒	Formule 4D associée à la colonne
typeDonnées	Entier long	⇒	Type de résultat de la formule

Description

La commande **LISTBOX FIXER FORMULE COLONNE** permet de modifier la *formule* associée à la colonne de list box désignée par les paramètres *objet* et *. Les formules ne peuvent être utilisées que lorsque la propriété “Source de données” de la list box est **Sélection courante** ou **Sélection temporaire**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d’objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Ce paramètre doit désigner une colonne de la listbox.

Le paramètre *formule* peut contenir toute expression valide, soit :

- une instruction,
- une formule générée à l’aide de l’éditeur de formules,
- un appel à une commande 4D,
- un appel à une méthode projet.

Au moment de l’appel de la commande, la formule est analysée puis exécutée.

Note : Utilisez la commande **Nom commande** afin de définir des formules indépendantes de la langue de l’application (lorsqu’elles font appel à des commandes 4D).

Le paramètre *typeDonnées* permet de désigner le type des données issues de l’exécution de la formule. Vous devez passer dans ce paramètre une des constantes du thème **Types champs et variables**. Si le résultat de la formule ne correspond pas au type de données attendu, une erreur est générée.

LISTBOX FIXER GRILLE

LISTBOX FIXER GRILLE ({* ;} objet ; horizontal ; vertical)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
horizontal	Booléen	⇒	Vrai=montrer, Faux=cacher
vertical	Booléen	⇒	Vrai=montrer, Faux=cacher

Description

La commande **LISTBOX FIXER GRILLE** permet d'afficher ou de masquer les traits horizontaux et/ou verticaux composant la grille de l'objet list box désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Passez dans les paramètres *horizontal* et *vertical* des valeurs booléennes indiquant si les traits correspondants doivent être affichés (**Vrai**) ou cachés (**Faux**). Par défaut, la grille est affichée.

LISTBOX FIXER HAUTEUR ENTETES

LISTBOX FIXER HAUTEUR ENTETES ({* ;} objet ; hauteur {; unité})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis
hauteur	Entier long	⇒	Hauteur de la ligne
unité	Entier long	⇒	Unité de la valeur de hauteur : 0 ou omis = pixels, 1 = lignes

Description

La commande **LISTBOX FIXER HAUTEUR ENTETES** permet de modifier par programmation la hauteur de la ligne d'en-tête de la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Vous pouvez désigner indifféremment la list box ou tout en-tête de la list box.

Passez dans le paramètre *hauteur* la hauteur à définir. Par défaut, si vous omettez le paramètre *unité*, cette hauteur est exprimée en pixels. Pour modifier l'unité, vous pouvez passer dans le paramètre *unité* l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk lignes	Entier long	1	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police.
lk pixels	Entier long	0	La hauteur est un nombre de pixels (défaut)

Les en-têtes doivent respecter une hauteur minimale, définie par le système d'exploitation. Cette hauteur est de 24 pixels sous Windows et 17 pixels sous Mac OS. Si vous passez une hauteur inférieure, la hauteur minimale est appliquée.

Note : Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au manuel *Mode Développement*.

LISTBOX FIXER HAUTEUR LIGNE

LISTBOX FIXER HAUTEUR LIGNE ({* ;} objet ; ligne ; hauteur)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ligne	Entier long	⇒	Ligne de la List box dont la hauteur doit être fixée.
hauteur	Entier long	⇒	Hauteur de la ligne de la List box

4D View Pro

Cette commande nécessite une licence 4D View Pro. Si cette licence n'est pas présente, une erreur est affichée dans la list box à l'exécution du formulaire. Pour plus d'informations, veuillez vous reporter à la section **4D View Pro**.

Description

La commande **LISTBOX FIXER HAUTEUR LIGNE** permet de modifier la hauteur de la ligne spécifiée dans le paramètre *ligne* dans la List box désignée par les paramètres *objet* et éventuellement *.

Si vous passez le paramètre optionnel *, vous précisez que le paramètre *objet* est un nom d'objet (chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous passez une référence de variable à la place d'une chaîne. Pour plus d'information sur les noms d'objet, reportez-vous à la section **Objets de formulaires**.

Si la *ligne* spécifiée n'existe pas dans la List box, la commande ne fait rien.

L'unité utilisée par *hauteur* correspond à celle définie globalement pour les lignes de la List box, soit dans la Liste des propriétés, soit par un appel antérieur à la commande **LISTBOX FIXER HAUTEUR LIGNES**.

La commande **LISTBOX FIXER HAUTEUR LIGNE** modifie le tableau de hauteur de lignes spécifié dans la Liste des propriétés, le cas échéant (cf. section **Tableau hauteurs de lignes** dans le manuel *Mode Développement*. Sinon, la commande crée dynamiquement un tableau de hauteurs de lignes. Utiliser cette commande pour définir individuellement les hauteurs de lignes produit le même résultat qu'utiliser un tableau de hauteurs de lignes ; toutefois, remplir un tableau de hauteurs de lignes est plus rapide qu'appeler cette commande dans une boucle pour fixer les hauteurs de lignes une par une.

Important : Si la commande globale **LISTBOX FIXER HAUTEUR LIGNES** est appelée par la suite avec une unité différente de celle définie précédemment, la valeur par défaut de cette commande remplacera et réinitialisera toute hauteur de ligne définie à l'aide de **LISTBOX FIXER HAUTEUR LIGNE** (voir exemple 2).

Exemple 1

Vous souhaitez modifier la hauteur de quelques lignes de la list box suivante :

RowNum	Countries	Population
1	Luxembourg	502 202
2	Latvia	1 973 700
3	Kuwait	4 044 500
4	Croatia	4 284 889
5	Denmark	5 699 220
6	Nicaragua	6 071 045
7	Serbia	7 306 677
8	Honduras	8 249 574
9	Austria	8 572 895
10	Hungary	10 005 000
11	Czech Republic	10 674 947

Si vous exécutez ce code :

```
 //unité courante en pixels
LISTBOX FIXER HAUTEUR LIGNE(*;"listboxname";3;40) //Kuwait
LISTBOX FIXER HAUTEUR LIGNE(*;"listboxname";7;14) //Serbia
```

... vous obtenez le résultat suivant :

RowNum	Countries	Population
1	Luxembourg	502 202
2	Latvia	1 973 700
3	Kuwait	4 044 500
4	Croatia	4 284 889
5	Denmark	5 699 220
6	Nicaragua	6 071 045
7	Serbia	7 306 677
8	Honduras	8 249 574
9	Austria	8 572 895
10	Hungary	10 005 000
11	Czech Republic	10 674 947

Exemple 2

Vous devez fixer une hauteur de ligne par défaut puis mettre en place individuellement des hauteurs de lignes spécifiques pour certaines d'entre elles en utilisant la commande **LISTBOX FIXER HAUTEUR LIGNE** :

```
LISTBOX FIXER HAUTEUR LIGNES(*;"listboxname";25;lk pixels) // la hauteur est globalement fixée à 25 pixels

LISTBOX FIXER HAUTEUR LIGNE(*;"listboxname";1;30) // ligne 1: 30 pixels
LISTBOX FIXER HAUTEUR LIGNE(*;"listboxname";5;40) // ligne 5: 40 pixels
LISTBOX FIXER HAUTEUR LIGNE(*;"listboxname";11;50) // ligne 11: 50 pixels
```

Par la suite, si le code suivant est exécuté :

```
LISTBOX FIXER HAUTEUR LIGNES(*;"listboxname";18;lk pixels)
```

… la hauteur des lignes est globalement fixée à 18 pixels ; toutefois, étant donné que l'unité n'a pas changé, les lignes 1, 5 et 11 garderont leur hauteur personnalisée, à savoir, 30, 40 et 50 pixels tel que défini ci-dessus par la commande **LISTBOX FIXER HAUTEUR LIGNE**.

En revanche, si le code suivant est exécuté :

```
LISTBOX FIXER HAUTEUR LIGNES(*;"listboxname";2;lk lignes)
```

… alors les lignes 1, 5 et 11 sont réinitialisées à la valeur globale par défaut mise en place par la commande **LISTBOX FIXER HAUTEUR LIGNES** (c'est-à-dire 2 lignes) car l'unité est passée de "pixels" à "lignes". Comme il n'y a pas de conversion automatique, le changement d'unité aboutit toujours à la réinitialisation des hauteurs de lignes avec la nouvelle valeur par défaut.

LISTBOX FIXER HAUTEUR LIGNES

LISTBOX FIXER HAUTEUR LIGNES ({* ;} objet ; hauteur {; unité})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
hauteur	Entier long	⇒	Hauteur de ligne
unité	Entier long	⇒	Unité de la valeur de hauteur : 0 ou omis = pixels, 1 = lignes

Description

La commande **LISTBOX FIXER HAUTEUR LIGNES** permet de modifier par programmation la hauteur des lignes de l'objet list box désigné par les paramètres *objet* et *.

Si vous passez le paramètre facultatif *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets (Formulaires)**.

Par défaut, si vous omettez le paramètre *unité*, la hauteur est exprimée en pixels. Pour modifier l'unité, vous pouvez passer dans le paramètre *unité* l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk lignes	Entier long	1	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police.
lk pixels	Entier long	0	La hauteur est un nombre de pixels (défaut)

Note : Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au manuel *Mode Développement*.

LISTBOX FIXER HAUTEUR PIEDS

LISTBOX FIXER HAUTEUR PIEDS ({* ;} objet ; hauteur {; unité})

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
hauteur	Entier long	➡	Hauteur de la ligne
unité	Entier long	➡	Unité de la valeur de hauteur : 0 ou omis = pixels, 1 = lignes

Description

La commande **LISTBOX FIXER HAUTEUR PIEDS** permet de modifier par programmation la hauteur de la ligne de pied de la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Vous pouvez désigner indifféremment la list box ou tout pied de la list box.

Passez dans le paramètre *hauteur* la hauteur à définir. Par défaut, si vous omettez le paramètre *unité*, cette hauteur est exprimée en pixels. Pour modifier l'unité, vous pouvez passer dans le paramètre *unité* l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk lignes	Entier long	1	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police.
lk pixels	Entier long	0	La hauteur est un nombre de pixels (défaut)

Note : Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au manuel *Mode Développement*.

LISTBOX FIXER HIERARCHIE

LISTBOX FIXER HIERARCHIE ({* ;} objet ; hiérarchique {; hiérarchie})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
hiérarchique	Booléen	⇒	Vrai = list box hiérarchique, Faux = list box non hiérarchique
hiérarchie	Tableau pointeur	⇒	Tableau de pointeurs

Description

La commande **LISTBOX FIXER HIERARCHIE** vous permet de configurer l'objet list box désigné par les paramètres *objet* et * en mode hiérarchique ou non.

Note : Cette commande fonctionne uniquement avec les list box basées sur des tableaux. Si elle est utilisée avec une list box basée sur des sélections, elle ne fait rien.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Le paramètre booléen *hiérarchique* vous permet de définir le mode de la list box :

- si vous passez Vrai, la list box est affichée en mode hiérarchique,
- si vous passez Faux, la list box est affichée en mode non hiérarchique (mode tableau standard).

Lorsque vous passez une list box en mode hiérarchique, certaines propriétés sont automatiquement restreintes. Pour plus d'informations, reportez-vous à la section **Gestion des List box hiérarchiques**.

Le paramètre *hiérarchie* vous permet de désigner les tableaux de la list box à utiliser pour construire la hiérarchie (cf. exemple).
Si vous affichez la list box en mode hiérarchique et omettez ce paramètre :

- si la list box est déjà en mode hiérarchique, la commande ne fait rien.
- si la list box est en mode non hiérarchique et n'a jamais été déclarée hiérarchique, le premier tableau est utilisé comme hiérarchie par défaut.
- si la list box est en mode non hiérarchique mais avait été déclarée hiérarchique précédemment, la dernière hiérarchie est rétablie.

Exemple

Définition des tableaux tPays, tRegion et tVille comme hiérarchie d'une list box :

```
TABLEAU POINTEUR($TabHierarch;3)
$TabHierarch{1}:=->tPays `Premier niveau de rupture
$TabHierarch{2}:=->tRegion `Deuxième niveau de rupture
$TabHierarch{3}:=->tVille `Troisième niveau de rupture
LISTBOX FIXER HIERARCHIE(*;"mylistbox";Vrai;$TabHierarch)
```

LISTBOX FIXER LARGEUR COLONNE

LISTBOX FIXER LARGEUR COLONNE ({* ;} objet ; largeur {; largeurMini {; largeurMaxi}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
largeur	Entier long	⇒	Largeur de colonne (en pixels)
largeurMini	Entier long	⇒	Largeur minimale de colonne (en pixels)
largeurMaxi	Entier long	⇒	Largeur maximale de colonne (en pixels)

Description

La commande **LISTBOX FIXER LARGEUR COLONNE** permet de modifier par programmation la largeur d'une ou de toutes les colonne(s) de l'objet (list box, colonne ou en-tête) désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Passez dans le paramètre *largeur* la nouvelle largeur (en pixels) de l'objet.

- Si *objet* désigne l'objet list box, toutes les colonnes de la list box sont redimensionnées.
- Si *objet* désigne une colonne ou un en-tête de colonne, seule la colonne désignée est redimensionnée.

Les paramètres optionnels *largeurMini* et *largeurMaxi* permettent de fixer des limites au redimensionnement manuel de la colonne. Vous pouvez passer dans *largeurMini* et *largeurMaxi* respectivement des valeurs de largeur minimale et maximale, exprimées en pixels. Si vous souhaitez que l'utilisateur ne puisse pas redimensionner la colonne, il suffit de passer la même valeur dans *largeur*, *largeurMini* et *largeurMaxi*.

LISTBOX FIXER STYLE LIGNE

LISTBOX FIXER STYLE LIGNE ({* ;} objet ; ligne ; style)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ligne	Entier long	⇒	Numéro de ligne
style	Entier long	⇒	Style de police

Description

Note : Cette commande fonctionne uniquement avec les list box de type tableau.

La commande **LISTBOX FIXER STYLE LIGNE** vous permet de définir un style de police pour une ligne ou une cellule de la list box tableau désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Vous pouvez désigner comme paramètre *objet* une list box ou une colonne de list box :

- si *objet* désigne une list box, la commande s'applique à la ligne
- si *objet* désigne une colonne, la commande s'applique à la cellule située à l'intersection colonne/ligne

Passez dans *ligne* le numéro de la ligne à laquelle le nouveau style doit être appliqué.

Note : La commande ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.

Passez dans *style* une valeur de style. Vous devez utiliser une ou une combinaison de constante(s) du thème **Styles de caractères** :

Constante	Type	Valeur
Gras	Entier long	1
Italique	Entier long	2
Normal	Entier long	0
Souligné	Entier long	4

Si un tableau de styles de police a été associé à la list box ou à la colonne, seul l'élément correspondant à la ligne sera modifié. Autrement dit, dans ce cas, l'exécution de la commande produit le même effet que la modification d'un élément du tableau de styles de police.
Si aucun tableau de styles de police n'a été associé à la list box ou à la colonne, il est créé dynamiquement lors de l'appel de la commande. Vous pourrez y accéder à l'aide de la commande **LISTBOX Lire tableau**.

Si des propriétés de style contradictoires sont définies pour la colonne ou la list box, un ordre de priorité est appliqué. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Note : Comme les tableaux de style des colonnes ont priorité sur le tableau de style de la list box, la commande, si elle est appliquée à une list box, n'aura d'effet que si aucun tableau de style n'a été affecté aux colonnes.

Exemple

Soit une list box tableau ayant les caractéristiques suivantes :

- un tableau de styles de police est associé à la list box (*ArrGlobalStyle*)
- un tableau de styles de police est associé à la colonne 5 (*ArrCol5Style*)
- les autres colonnes n'ont pas de tableau de style

```
LISTBOX FIXER STYLE LIGNE(*;"Col5";3;Gras)
    // équivaut à ArrCol5Style{3}:=Gras
```

text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	**text**	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
LISTBOX FIXER STYLE LIGNE(*;"List Box";3;Italique+Souligné)
    // équivaut à ArrGlobalStyle{3}:=Italique+Souligné
```

text	text	text	text	text	text
text	text	text	text	text	text
text	*text*	*text*	*text*	**text**	*text*
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

Après la deuxième instruction, toutes les cellules de la troisième ligne passent en italique et souligné sauf celle de la colonne 5, qui reste en gras uniquement (les tableaux de colonnes sont prioritaires sur les tableaux de list box).

LISTBOX FIXER TABLE SOURCE

LISTBOX FIXER TABLE SOURCE ({* ;} objet ; numTable | tempo {; nomSurlignage})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
numTable | tempo	Entier long, Chaîne	⇒	Numéro de la table de laquelle utiliser la sélection courante ou Nom de la sélection temporaire à utiliser
nomSurlignage	Chaîne	⇒	Nom de l'ensemble de surlignage

Description

La commande **LISTBOX FIXER TABLE SOURCE** vous permet de modifier la source des données affichées dans la list box désignée par les paramètres *** et *objet*.

Note : Cette commande ne peut être utilisée que lorsque la propriété "Source de données" de la list box est **Sélection courante** ou **Sélection temporaire** (pour plus d'informations sur ce point, reportez-vous à la section **Gestion programmée des objets de type List box**). Elle ne fait rien si vous l'utilisez avec une list box associée à des tableaux.

Si vous passez le paramètre optionnel ***, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Si vous passez un numéro de table comme paramètre *numTable*, la list box sera remplie avec les données des enregistrements de la sélection courante de la table.
Si vous passez un nom de sélection temporaire comme paramètre *tempo*, la list box sera remplie avec les données des enregistrements appartenant à la sélection temporaire.

Le paramètre optionnel *nomSurlignage* vous permet d'associer un ensemble de surlignage à la list box. L'ensemble de surlignage est utilisé pour gérer le surlignage des enregistrements par l'utilisateur dans la list box.

Si la list box contenait déjà des colonnes, leur contenu est mis à jour à l'issue de l'exécution de la commande.

Note : Pour des raisons d'optimisation, cette commande est traitée de manière asynchrone, c'est-à-dire que le changement de source de la listbox n'est effectif qu'à l'issue de l'exécution complète de la méthode dans laquelle la commande est appelée.

LISTBOX FIXER TABLEAU

LISTBOX FIXER TABLEAU ({* ;} objet ; typeTab ; ptrTab)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
typeTab	Entier long	⇒	Type de tableau
ptrTab	Pointeur	⇒	Tableau à associer à la propriété

Description

Note : Cette commande fonctionne uniquement avec les list box de type tableau.

La commande **LISTBOX FIXER TABLEAU** vous permet d'associer un tableau de type *typeTab* à la list box ou à la colonne de list box désignée par les paramètres *objet* et *.

Note : Des tableaux de style, de couleur, de couleur de fond ou de contrôle des lignes peuvent également être associés aux list box de type tableau via la Liste des propriétés en mode Développement.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Vous pouvez désigner comme paramètre *objet* une list box ou une colonne de list box.

Passez dans *typeTab* le type de tableau à associer à la list box ou à la colonne. Vous pouvez utiliser une des constantes suivantes du thème "**List box**" :

Constante	Type	Valeur	Comment
lk tableau contrôle	Entier long	3	
lk tableau couleur de fond	Entier long	1	
lk tableau couleur de police	Entier long	0	
lk tableau hauteurs lignes	Entier long	4	(Licence 4D View Pro requise)
lk tableau style	Entier long	2	

Passez dans le paramètre *ptrTab* un pointeur vers le tableau à utiliser pour prendre en charge le type de propriété.

Exemple 1

Vous souhaitez réutiliser le tableau de couleurs de police de la colonne 4 pour la colonne 10 :

```
    // récupérer un pointeur vers le tableau de la colonne 4
$Pointer:=LISTBOX Lire tableau(*;"Col4";lk tableau couleur de police)
    // vérification qu'il existe
Si(Non(Nil($Pointer)))
         //report sur la colonne 10
      LISTBOX FIXER TABLEAU(*;"Col10";lk tableau couleur de police;$Pointer)
Fin de si
```

Exemple 2

Vous voulez associer un tableau de hauteurs de ligne à une list box :

```
LISTBOX FIXER TABLEAU(*;"LB";lk tableau hauteurs lignes;->RowHeightArray)
```

Note : La propriété de list box **Tableau hauteurs lignes** nécessite la licence 4D View Pro. Pour plus d'informations, veuillez vous reporter à la section **4D View Pro**.

LISTBOX INSERER COLONNE

LISTBOX INSERER COLONNE ({* ;} objet ; positionCol ; nomCol ; variableCol ; nomEnTête ; variableEntête {; nomPied ; variablePied})

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionCol	Entier long	➡	Emplacement de la colonne à insérer
nomCol	Chaîne	➡	Nom d'objet de la colonne
variableCol	Tableau, Champ, Variable, Pointeur nil	➡	Nom de la variable tableau de la colonne ou champ ou variable
nomEnTête	Chaîne	➡	Nom d'objet de l'en-tête de la colonne
variableEntête	Variable entier, Pointeur nil	➡	Variable d'en-tête de la colonne
nomPied	Chaîne	➡	Nom d'objet du pied de la colonne
variablePied	Variable, Pointeur nil	➡	Variable du pied de la colonne

Description

La commande **LISTBOX INSERER COLONNE** insère une colonne dans la list box désignée par les paramètres *objet* et *.

Note : Cette commande ne fait rien si elle est appliquée à la première colonne d'une list box affichée en mode hiérarchique.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

La nouvelle colonne est insérée juste avant la colonne désignée par le paramètre *positionCol*. Si le paramètre *positionCol* est supérieur au nombre total de colonnes, la colonne est ajoutée après la dernière colonne.

Passez dans les paramètres *nomCol* et *variableCol* le nom d'objet et la variable de la colonne insérée.

- Dans le cadre d'une list box de type tableau, le nom de la variable correspond au nom du tableau dont le contenu sera affiché dans la colonne. Vous pouvez passer un pointeur Nil (->[]) si vous utilisez la commande dans un contexte dynamique à l'exécution du formulaire (cf. ci-dessous).
- Dans le cadre d'une list box de type sélection, vous pouvez passer un champ ou une variable dans le paramètre *variableCol*. Le contenu de la colonne sera alors la valeur du champ ou de la variable, évaluée pour chaque enregistrement de la sélection associée à la list box. Ce type de contenu ne peut être utilisé que lorsque la propriété "Source de données" de la list box est Sélection courante ou Sélection temporaire (cf. section **Gestion programmée des objets de type List box**). Vous pouvez utiliser des champs ou des variables de type chaîne, numérique, Date, Heure, Image et Booléen.

Dans le contexte de list box basées sur des sélections, **LISTBOX INSERER COLONNE** permet d'insérer des éléments simples (champs ou variables). Si vous souhaitez manipuler des expressions plus complexes (telles que des formules ou des méthodes), vous devez utiliser la commande **LISTBOX INSERER COLONNE FORMULE**.

Note : Il n'est pas possible de combiner dans une même list box des colonnes de type tableau (source de données tableaux) et des colonnes de type champ ou variable (source de données sélection).

Passez dans les paramètres *nomEntête* et *variableEntête* le nom d'objet et la variable de l'en-tête de la colonne insérée.

Vous pouvez également passer dans les paramètres *nomPied* et *variablePied* le nom d'objet et la variable du pied de la colonne insérée.

Note : Les noms d'objets doivent être uniques dans un formulaire. Vous devez veiller à ce que les noms passés dans les paramètres *nomCol*, *nomEntête* et *nomPied* ne soient pas déjà utilisés. Sinon, la colonne n'est pas créée et une erreur est générée.

Insertion dynamique

A compter de 4D v14 R3, vous pouvez utiliser cette commande pour insérer dynamiquement des colonnes dans les list box à l'exécution du formulaire, 4D prenant automatiquement en charge les définitions de variables nécessaires (colonne, pied et en-tête).

Pour cela, **LISTBOX INSERER COLONNE** accepte un pointeur **Nil** (**->[]**) comme valeur pour les paramètres *variableCol* (list box de type tableau uniquement), *variableEntête* et *variablePied*. Dans ce cas, 4D va créer dynamiquement les variables requises lors de l'exécution de la commande (pour plus d'informations, reportez-vous à la section **Variables dynamiques**).

A noter que les variables d'en-tête et de pied sont toujours créées avec un type spécifique (respectivement entier long et texte). A l'inverse, les variables de colonne ne peuvent pas être typées à la création car les list box acceptent différents types de tableaux pour ces variables (tableau texte, tableau entier, etc.). Vous devez donc définir manuellement le type du tableau (cf. exemple 3). Il est important d'effectuer ce typage avant d'appeler des commandes telles que **LISTBOX INSERER LIGNES** pour insérer des nouveaux éléments dans le tableau. Ou bien, il est possible d'utiliser **AJOUTER A TABLEAU** pour à la fois typer le tableau et insérer des éléments.

Exemple 1

Nous souhaitons ajouter une colonne à la fin de la list box :

```
C_ENTIER LONG(NomVarHeader;$Der;$NbEnr)
TOUT SELECTIONNER([Table 1])
$NbEnr:=Enregistrements dans table([Table 1])
TABLEAU IMAGE(tabImage;$NbEnr)

$Der:=LISTBOX Lire nombre colonnes(*;"ListBox1")+1
LISTBOX INSERER COLONNE(*;"ListBox1";$Der;"ColumnPicture";tabImage;"HeaderPicture";NomVarHeader)
```

Exemple 2

Nous souhaitons ajouter une colonne à la droite de la list box et lui associer les valeurs du champ [Envois]Frais :

```
$der:=LISTBOX Lire nombre colonnes(*;"ListBox1")+1
```

```
LISTBOX INSERER COLONNE(*;"ListBox1";$der;"ColChamp";[Envois]Frais;"NomEntete";VarEntete)
```

Exemple 3

Vous souhaitez insérer dynamiquement une colonne dans une list box de type tableau et définir son en-tête :

```
C_POINTEUR($NilPtr)
LISTBOX INSERER COLONNE(*;"MyListBox";1;"MyNewColumn";$NilPtr;"MyNewHeader";$NilPtr)
ColPtr:=OBJET Lire pointeur(Objet nommé;"MyNewColumn")
TABLEAU TEXTE(ColPtr->;10)
    //Définition de l'en-tête
headprt:=OBJET Lire pointeur(Objet nommé;"MyNewHeader")
OBJET FIXER TITRE(headprt->;"Entête inséré")
```

LISTBOX INSERER COLONNE FORMULE

LISTBOX INSERER COLONNE FORMULE ({* ;} objet ; positionCol ; nomCol ; formule ; typeDonnées ; nomEnTête ; variableEntête {; nomPied ; variablePied})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionCol	Entier long	⇒	Emplacement de la colonne à insérer
nomCol	Chaîne	⇒	Nom d'objet de la colonne
formule	Chaîne	⇒	Formule 4D associée à la colonne
typeDonnées	Entier long	⇒	Type de résultat de la formule
nomEnTête	Chaîne	⇒	Nom d'objet de l'en-tête de la colonne
variableEntête	Variable entier, Pointeur nil	⇒	Variable d'en-tête de la colonne
nomPied	Chaîne	⇒	Nom d'objet du pied de la colonne
variablePied	Variable, Pointeur nil	⇒	Variable du pied de la colonne

Description

La commande **LISTBOX INSERER COLONNE FORMULE** insère une colonne dans la list box désignée par les paramètres *objet* et *.
La commande **LISTBOX INSERER COLONNE FORMULE** est semblable à la commande **LISTBOX INSERER COLONNE**, à la différence près qu'elle permet la saisie d'une formule comme contenu de la colonne.
Ce type de contenu ne peut être utilisé que lorsque la propriété "Source de données" de la list box est **Sélection courante** ou **Sélection temporaire** (pour plus d'informations sur ce point, reportez-vous à la section **Gestion programmée des objets de type List box**).

Note : Cette commande ne fait rien si elle est appliquée à la première colonne d'une list box affichée en mode hiérarchique.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

La nouvelle colonne est insérée juste avant la colonne désignée par le paramètre *positionCol*. Si le paramètre *positionCol* est supérieur au nombre total de colonnes, la colonne est ajoutée après la dernière colonne.

Passez dans le paramètre *nomCol* le nom d'objet de la colonne insérée.

Le paramètre *formule* peut contenir toute expression valide, soit :

- une instruction,
- une formule générée à l'aide de l'éditeur de formules,
- un appel à une commande 4D,
- un appel à une méthode projet.

Au moment de l'appel de la commande, la *formule* est analysée puis exécutée.

Note : Utilisez la commande **Nom commande** afin de définir des formules indépendantes de la langue de l'application (lorsqu'elles font appel à des commandes 4D).

Le paramètre *typeDonnées* permet de désigner le type des données issues de l'exécution de la *formule*. Vous devez passer dans ce paramètre une des constantes du thème **Types champs et variables** suivantes :

Constante	Type	Valeur
Est un booléen	Entier long	6
Est un numérique	Entier long	1
Est un texte	Entier long	2
Est une date	Entier long	4
Est une heure	Entier long	11
Est une image	Entier long	3

Si le résultat de la *formule* ne correspond pas au type de données attendu, une erreur est générée.

Passez dans les paramètres *nomEntête* et *variableEntête* le nom d'objet et la variable de l'en-tête de la colonne insérée.

Vous pouvez également passer dans les paramètres *nomPied* et *variablePied* le nom d'objet et la variable du pied de la colonne insérée. Si vous omettez le paramètre *variablePied*, 4D utilisera une variable dynamique.

Note : Les noms d'objets doivent être uniques dans un formulaire. Vous devez veiller à ce que les noms passés dans les paramètres *nomCol*, *nomEntête* et *nomPied* ne soient pas déjà utilisés. Sinon, la colonne n'est pas créée et une erreur est générée.

Insertion dynamique

A compter de 4D v14 R3, vous pouvez utiliser cette commande pour insérer dynamiquement des colonnes dans les list box à l'exécution du formulaire, 4D prenant automatiquement en charge les définitions de variables nécessaires (pied et en-tête).

Pour cela, **LISTBOX INSERER COLONNE FORMULE** accepte un pointeur **Nil** (**->[]**) comme valeur pour les paramètres *variableEntête* et *variablePied*. Dans ce cas, 4D va créer dynamiquement les variables requises lors de l'exécution de la commande (pour plus d'informations, reportez-vous à la section **Variables dynamiques**).

A noter que les variables d'en-tête et de pied sont toujours créées avec un type spécifique (respectivement entier long et texte).

Exemple

Nous souhaitons ajouter une nouvelle colonne à la droite de la list box qui contiendra une formule calculant l'âge de l'employé :

```
vAge:="Date du jour-[Employés]DateNaissance)\365"
$der:=LISTBOX Lire nombre colonnes(*;"ListBox1")+1
LISTBOX INSERER COLONNE FORMULE(*;"ListBox1";$der;"ColFormule";vAge;Est un numérique;"Age";VarEntete)
```

LISTBOX INSERER LIGNES

LISTBOX INSERER LIGNES ({* ;} objet ; positionLigne {; nbLignes})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionLigne	Entier long	⇒	Emplacement de la ligne à insérer
nbLignes	Entier long	⇒	Nombre de lignes à insérer

Description

La commande **LISTBOX INSERER LIGNES** insère une ou plusieurs nouvelle(s) ligne(s) dans l'objet list box désigné par les paramètres *objet* et *.

Note : Cette commande fonctionne uniquement avec les list box basées sur des tableaux. Lorsque cette commande est utilisée avec une list box basée sur des sélections, elle ne fait rien et la variable système OK retourne 0.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Par défaut, si le paramètre *nbLignes* est omis, une seule ligne est insérée. Sinon, la commande insère le nombre de lignes défini dans ce paramètre.

La ou les ligne(s) est (sont) insérée(s) à l'emplacement défini par le paramètre *positionLigne* et est (sont) automatiquement ajoutée(s) à cet emplacement dans tous les tableaux composant la list box, quel que soit leur type, y compris dans les tableaux (colonnes) masqués.

Si le paramètre *positionLigne* est supérieur au nombre de lignes des tableaux de la list box, la commande effectue l'ajout à la fin de chaque tableau. S'il est égal à 0, la commande effectue l'ajout au début de chaque tableau. S'il contient une valeur négative, la commande ne fait rien.

LISTBOX Lire calcul pied

LISTBOX Lire calcul pied ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	⇐	Type de calcul

Description

La commande **LISTBOX Lire calcul pied** retourne le type de calcul associé à la zone de pied de list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Le paramètre *objet* peut désigner :

- la variable ou le nom d'une zone de pied. Dans ce cas, la commande retourne le calcul associé à cette zone.
- la variable ou le nom d'une colonne de list box. Dans ce cas, la commande retourne le calcul associé à la zone de pied de cette colonne.

Vous pouvez comparer la valeur retournée aux constantes du thème **List box pied calcul** (cf. commande **LISTBOX FIXER CALCUL PIED**).

LISTBOX Lire colonnes statiques

LISTBOX Lire colonnes statiques ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	⇐	Nombre de colonnes statiques

Description

La commande **LISTBOX Lire colonnes statiques** retourne le nombre de colonnes statiques dans la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Les colonnes statiques peuvent avoir été définies via la Liste des propriétés ou à l'aide de la commande **LISTBOX FIXER COLONNES STATIQUES**.

Si une colonne a été insérée ou supprimée par programmation à l'intérieur d'un ensemble de colonnes statiques, le nombre de colonnes retourné par cette commande tient compte de cette modification. En revanche, la commande ne tient pas compte du statut visible/invisible des colonnes.

Note : Les colonnes statiques et les colonnes verrouillées sont deux fonctionnalités indépendantes. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

LISTBOX Lire colonnes verrouillees

LISTBOX Lire colonnes verrouillees ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	⇐	Nombre de colonnes verrouillées

Description

La commande **LISTBOX Lire colonnes verrouillees** retourne le nombre de colonnes verrouillées dans la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Les colonnes peuvent avoir été verrouillées via la Liste des propriétés ou à l'aide de la commande **LISTBOX FIXER COLONNES VERROUILLEES**. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Si une colonne a été insérée ou supprimée par programmation à l'intérieur de la zone de verrouillage, le nombre de colonnes retourné par cette commande tient compte de cette modification. Par exemple, si vous supprimez une colonne verrouillée, le nombre de colonnes verrouillées sera diminué de 1. De même, si une colonne est insérée par programmation dans la zone de verrouillage, elle est automatiquement verrouillée et le nombre de colonnes verrouillées sera augmenté de 1.

En revanche, la commande ne tient pas compte du statut visible/invisible des colonnes.

LISTBOX LIRE COORDONNEES CELLULE

LISTBOX LIRE COORDONNEES CELLULE ({* ;} objet ; colonne ; ligne ; gauche ; haut ; droite ; bas)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
colonne	Entier long	⇒	Numéro de colonne
ligne	Entier long	⇒	Numéro de ligne
gauche	Entier long	⇐	Coordonnée gauche de l'objet
haut	Entier long	⇐	Coordonnée supérieure de l'objet
droite	Entier long	⇐	Coordonnée droite de l'objet
bas	Entier long	⇐	Coordonnée inférieure de l'objet

Description

La commande **LISTBOX LIRE COORDONNEES CELLULE** retourne dans les variables ou champs *gauche*, *haut*, *droite* et *bas* les coordonnées (en points) de la cellule désignée par les paramètres *colonne* et *ligne* dans la list box définie par * et *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne de caractères). Si vous ne passez pas le paramètre *, vous indiquez que l'objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable de type objet uniquement).

Par cohérence avec la commande **OBJET LIRE COORDONNEES**, le point d'origine est le coin supérieur gauche du formulaire qui contient la cellule. Aussi, les coordonnées retournées sont théoriques ; elles tiennent compte du défilement éventuel de la list box avant son affichage à l'intérieur de son cadre. En résultat, la cellule peut ne pas être visible (ou être visible seulement en partie) à ses coordonnées, et ces coordonnées peuvent se situer au-delà des limites du formulaire (voire être négatives). Pour savoir si la cellule est visible (et quelle partie) vous devez comparer les coordonnées retournées avec celles de la list box elle-même, en tenant compte des règles suivantes :

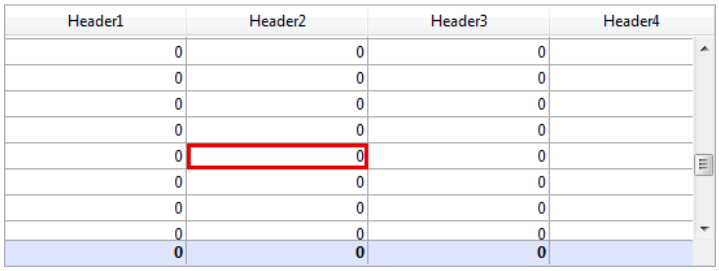
- Les limites des cellules dépendent des coordonnées de leur list box parente (telles que retournées par la commande **OBJET LIRE COORDONNEES** pour la list box).
- Les sous-objets en-tête et pied sont affichés au-dessus du contenu des colonnes : lorsque les coordonnées d'une cellule coupent celles d'une ligne d'en-tête ou de pied, la cellule n'est pas affichée à l'emplacement de l'intersection.
- Les éléments des colonnes verrouillées sont affichés au-dessus des éléments des colonnes défilables : lorsque les coordonnées d'un élément d'une colonne défilable croisent celles d'un élément d'une colonne verrouillée, l'élément défilable n'est pas affiché à l'emplacement de l'intersection.

Pour plus d'informations, veuillez vous reporter à la description de la commande **OBJET LIRE COORDONNEES**.

Exemple

Vous souhaitez afficher un rectangle rouge autour de la cellule sélectionnée dans une list box :

```
OBJET FIXER VISIBLE(*;"RedRect";Faux) //initialiser un rectangle rouge
 //le rectangle est déjà défini quelque part dans le formulaire
LISTBOX LIRE POSITION CELLULE(*;"LB1";$col;$raw)
LISTBOX LIRE COORDONNEES CELLULE(*;"LB1";$col;$raw;$x1;$y1;$x2;$y2)
OBJET FIXER VISIBLE(*;"RedRect";Vrai)
OBJET FIXER COORDONNEES(*;"RedRect";$x1;$y1;$x2;$y2)
```



LISTBOX LIRE COULEUR GRILLE

LISTBOX LIRE COULEUR GRILLE ({* ;} objet ; couleurH ; couleurV)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
couleurH	Entier long	⇐	Valeur de couleur RVB pour les traits horizontaux
couleurV	Entier long	⇐	Valeur de couleur RVB pour les traits verticaux

Description

La commande **LISTBOX LIRE COULEUR GRILLE** retourne la couleur des lignes horizontales et verticales composant la grille de l'objet list box désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

La commande retourne dans les paramètres *couleurH* et *couleurV* des valeurs de couleurs RVB. Pour plus d'informations sur les couleurs RVB, reportez-vous à la description de la commande **OBJET FIXER COULEURS RVB**.

LISTBOX Lire couleur ligne

LISTBOX Lire couleur ligne ({* ;} objet ; ligne {; typeCouleur}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ligne	Entier long	⇒	Numéro de ligne
typeCouleur	Entier long	⇒	lk couleur de police (défaut) ou lk couleur de fond
Résultat	Entier long	⇐	Valeur de couleur

Description

Note : Cette commande fonctionne uniquement avec les list box de type tableau.

La commande **LISTBOX Lire couleur ligne** retourne la couleur d'une ligne ou d'une cellule de la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Vous pouvez désigner comme paramètre *objet* une list box ou une colonne de list box :

- si *objet* désigne une list box, la commande retourne la couleur de la ligne
- si *objet* désigne une colonne, la commande retourne la couleur de la cellule

Passez dans *ligne* le numéro de la ligne dont vous souhaitez obtenir la couleur.

Note : La commande ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.

Passez la constante lk couleur de fond ou lk couleur de police (thème "**List box**") dans le paramètre *typeCouleur* selon que vous souhaitez connaître la couleur de fond ou la couleur de police de la ligne. Si vous omettez ce paramètre, la couleur de police est retournée.

Attention, une couleur affectée à une ligne n'est pas forcément affichée dans toutes les cellules de ligne (cf. exemple). Si des valeurs de couleur contradictoires sont définies via les propriétés de la list box ou de la colonne, un ordre de priorité est appliqué. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Exemple

Soit la list box suivante :

text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
vCoul:=LISTBOX Lire couleur ligne(*;"Col5";3)
vCoul2:=LISTBOX Lire couleur ligne(*;"List Box";3)
vCoul3:=LISTBOX Lire couleur ligne(*;"List Box";lk couleur de fond)
    // vCoul contient 0xFFFF00 (jaune)
    // vCoul2 contient 0x00FF (bleu)
    // vCoul3 contient 0x00FF0000 (rouge)
```

LISTBOX Lire formule colonne

LISTBOX Lire formule colonne ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Chaîne	⇐	Formule associée à la colonne

Description

La commande **LISTBOX Lire formule colonne** retourne la formule associée à la colonne de list box désignée par les paramètres *objet* et *. Les formules ne peuvent être utilisées que lorsque la propriété “Source de données” de la list box est **Sélection courante** ou **Sélection temporaire**. Si aucune formule n’est associée à la colonne, la commande retourne une chaîne vide.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d’objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Ce paramètre doit désigner une colonne de la listbox.

LISTBOX LIRE GRILLE

LISTBOX LIRE GRILLE ({* ;} objet ; horizontal ; vertical)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
horizontal	Booléen	⇐	Vrai = affichée, Faux = cachée
vertical	Booléen	⇐	Vrai = affichée, Faux = cachée

Description

La commande **LISTBOX LIRE GRILLE** retourne le statut affiché/masqué des lignes horizontales et/ou verticales composant la grille de l'objet list box désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

La commande retourne dans les paramètres *horizontal* et *vertical* la valeur **Vrai** ou **Faux** selon que si les lignes correspondantes sont affichées (Vrai) ou cachées (Faux). Par défaut, la grille est affichée.

LISTBOX Lire hauteur entetes

LISTBOX Lire hauteur entetes ({* ;} objet {; unité}) -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
unité	Entier long	➡	Unité de la valeur de hauteur : 0 ou omis = pixels, 1 = lignes
Résultat	Entier long	⬅	Hauteur de la ligne

Description

La commande **LISTBOX Lire hauteur entetes** retourne la hauteur de la ligne d'en-tête de la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Vous pouvez désigner indifféremment la list box ou tout en-tête de la list box.

Par défaut, si vous omettez le paramètre *unité*, la hauteur de ligne retournée est exprimée en pixels. Pour définir une autre unité, vous pouvez passer dans le paramètre *unité* l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk lignes	Entier long	1	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police.
lk pixels	Entier long	0	La hauteur est un nombre de pixels (défaut)

Note : Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au manuel *Mode Développement*.

LISTBOX Lire hauteur ligne

LISTBOX Lire hauteur ligne ({* ;} objet ; ligne) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ligne	Entier long	⇒	ligne de la List box dont on veut récupérer la hauteur
Résultat	Entier	⇐	Hauteur de la ligne

4D View Pro

Cette commande nécessite une licence 4D View Pro. Si cette licence n'est pas présente, une erreur est affichée dans la list box à l'exécution du formulaire. Pour plus d'informations, veuillez vous reporter à la section **4D View Pro**.

Description

La commande **LISTBOX Lire hauteur ligne** retourne la hauteur de la *ligne* spécifiée dans l'objet List box désigné en utilisant les paramètres *objet* et éventuellement *. La hauteur des lignes peut être définie globalement via la Liste des propriétés ou la commande **LISTBOX FIXER HAUTEUR LIGNES**, ou individuellement à l'aide de la commande **LISTBOX FIXER HAUTEUR LIGNE**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous passez une référence à une variable à la place d'une chaîne. Pour plus d'informations sur les noms d'objet, reportez vous à la section **Objets de formulaires**.

Si la *ligne* spécifiée n'existe pas, la commande retourne 0 (zéro).

La hauteur de la ligne est retournée dans l'unité définie globalement pour la List box, soit dans la Liste des propriétés, soit par un appel antérieur à la commande **LISTBOX FIXER HAUTEUR LIGNES**.

LISTBOX Lire hauteur lignes

LISTBOX Lire hauteur lignes ({* ;} objet {; unité}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
unité	Entier long	⇒	Unité de la valeur de hauteur : 0 ou omis = pixels, 1 = lignes
Résultat	Entier	⇐	Hauteur de ligne en pixels

Description

La commande **LISTBOX Lire hauteur lignes** retourne la hauteur courante des lignes de l'objet list box désigné par les paramètres *objet* et *. La valeur retournée correspond à la hauteur d'une seule ligne.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Par défaut, si vous omettez le paramètre *unité*, la hauteur de ligne retournée est exprimée en pixels. Pour définir une autre unité, vous pouvez passer dans le paramètre *unité* l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk lignes	Entier long	1	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police.
lk pixels	Entier long	0	La hauteur est un nombre de pixels (défaut)

Note : Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au manuel *Mode Développement*.

LISTBOX Lire hauteur pieds

LISTBOX Lire hauteur pieds ({* ;} objet {; unité}) -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
unité	Entier long	➡	Unité de la valeur de hauteur : 0 ou omis = pixels, 1 = lignes
Résultat	Entier long	⬅	Hauteur de la ligne

Description

La commande **LISTBOX Lire hauteur pieds** retourne la hauteur de la ligne de pied de la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Vous pouvez désigner indifféremment la list box ou tout pied de la list box.

Par défaut, si vous omettez le paramètre *unité*, la hauteur de ligne retournée est exprimée en pixels. Pour définir une autre unité, vous pouvez passer dans le paramètre *unité* l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk lignes	Entier long	1	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police.
lk pixels	Entier long	0	La hauteur est un nombre de pixels (défaut)

Note : Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au manuel *Mode Développement*.

LISTBOX LIRE HIERARCHIE

LISTBOX LIRE HIERARCHIE ({* ;} objet ; hiérarchique {; hiérarchie})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
hiérarchique	Booléen	⇐	Vrai = list box hiérarchique, Faux = list box non hiérarchique
hiérarchie	Tableau pointeur	⇐	Tableau de pointeurs

Description

La commande **LISTBOX LIRE HIERARCHIE** vous permet de connaître les propriétés hiérarchiques de l'objet list box désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Le paramètre booléen *hiérarchique* indique si list box est en mode hiérarchique ou non :

- si le paramètre retourne Vrai, la list box est en mode hiérarchique,
- si le paramètre retourne Faux, la list box est affichée en mode non hiérarchique (mode tableau standard).

Si la list box est en mode hiérarchique, la commande remplit le tableau *hiérarchie* avec des pointeurs vers les tableaux de la list box utilisés pour définir la hiérarchie.

Note : Si la list box est en mode non hiérarchique, la commande retourne dans le premier élément du tableau *hiérarchie* un pointeur vers le tableau de la première colonne de la list box.

LISTBOX Lire information

LISTBOX Lire information ({* ;} objet ; info) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
info	Entier long	⇒	Type d'information à lire
Résultat	Entier long	⇐	Valeur courante

Description

La commande **LISTBOX Lire information** permet de récupérer différentes informations concernant la taille et la visibilité des en-têtes, des pieds et des barres de défilement de l'objet list box désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Passez dans le paramètre *info* un code correspondant à l'information que vous souhaitez obtenir. Vous pouvez passer une valeur ou l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk affichage barre déf hor	Entier long	2	0=masquée, 1=affichée
lk affichage barre déf ver	Entier long	4	0=masquée, 1=affichée
lk affichage entête	Entier long	0	Propriété **Afficher en-têtes** S'applique à : List box Valeurs possibles : • lk non (0) : masqué • lk oui (1) : affiché
lk affichage pied	Entier long	8	Propriété **Afficher pieds** S'applique à : List box Valeurs possibles : • lk non (0) : masqué • lk oui (1) : affiché
lk hauteur barre déf hor	Entier long	3	Hauteur en pixels
lk hauteur entête	Entier long	1	Hauteur en pixels
lk hauteur pied	Entier long	9	Hauteur en pixels
lk largeur barre déf ver	Entier long	5	Largeur en pixels
lk position barre déf hor	Entier long	6	Position du curseur en pixels
lk position barre déf ver	Entier long	7	Position du curseur en pixels

- Les premières constantes sont utiles pour calculer la taille de la zone de list box affichée dans le formulaire.
- Lorsque vous utilisez la constante lk position barre déf hor ou la constante lk position barre déf ver, la commande retourne la position relative du curseur de défilement par rapport à son origine, c'est-à-dire la taille de la partie masquée de la fenêtre, exprimée en pixels. Par défaut, cette position correspond à 0. Combinée par exemple aux informations relatives à la hauteur des lignes, cette valeur permet de connaître le contenu affiché dans la list box.
- L'instruction **LISTBOX Lire information**(vLB;lk hauteur pied) retourne la même valeur que la commande **LISTBOX Lire hauteur pieds** lorsque les pieds sont affichés. Dans le cas contraire, **LISTBOX Lire information** retourne 0 alors que **LISTBOX Lire hauteur pieds** retourne toujours la hauteur, dans ce cas théorique, des pieds.

Exemple

Soit une list box contenant des lignes d'une hauteur de 20 pixels chacune. Vous exécutez l'instruction suivante :

```
$déf:=LISTBOX Lire information(*;"Listbox";lk position barre déf ver)
```

Si, par exemple, $déf retourne 200, vous pouvez en déduire que la 11e ligne est actuellement la première affichée dans la list box (200/20=10, donc 10 lignes sont masquées).

LISTBOX LIRE INFORMATION IMPRESSION

LISTBOX LIRE INFORMATION IMPRESSION ({* ;} objet ; sélecteur ; info)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Variable (si * omis)
sélecteur	Entier long	⇒	Information à obtenir
info	Entier long	⇐	Valeur courante

Description

La commande **LISTBOX LIRE INFORMATION IMPRESSION** retourne des informations courantes relatives à l'impression de l'objet list box désigné par les paramètres *objet* et *. Cette commande permet de contrôler l'impression du contenu de la list box.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Cette commande doit être appelée dans le contexte de l'impression d'une list box via la commande **Imprimer objet**. Hors de ce contexte, elle ne retourne pas de valeurs significatives.

Passez dans *sélecteur* une valeur indiquant l'information à connaître et dans *info* une variable de type numérique ou BLOB. Vous pouvez passer dans *sélecteur* une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk hauteur imprimée	Entier long	3	Retourne dans *info* la hauteur en pixels de l'objet effectivement imprimé (inclut les en-têtes, filets, etc.). Souvenez-vous que si le nombre de lignes à imprimer est inférieur à la "capacité" de la list box, sa hauteur est automatiquement réduite.
lk impression terminée	Entier long	2	Retourne dans *info* un booléen indiquant si la dernière ligne (visible) de la list box a été effectivement imprimée. Vrai = la ligne a été imprimée, Faux sinon.
lk nombre lignes imprimées	Entier long	1	Retourne dans *info* le nombre de lignes effectivement imprimées lors du dernier appel à la commande **Imprimer objet**. Ce nombre inclut les éventuelles lignes de ruptures ajoutées dans le cadre d'une list box hiérarchique. Par exemple, *info* vaut 10 si la list box contient 20 lignes et que les lignes impaires ont été masquées.
lk num dernière ligne impr	Entier long	0	Retourne dans *info* le numéro de la dernière ligne imprimée. Permet de connaître le numéro de la prochaine ligne à imprimer. Le numéro retourné peut être supérieur au nombre de lignes effectivement imprimées si la list box contient des lignes invisibles ou si la commande **OBJET FIXER DEFILEMENT** a été appelée. Par exemple, si les lignes 1, 18 et 20 ont été imprimées, *info* vaut 20.

Pour plus d'informations sur les principes d'impression des list box, reportez-vous au paragraphe **Gestion des impressions**.

Exemple 1

Impression jusqu'à ce que toutes les lignes soient imprimées :

```
OUVRIR TACHE IMPRESSION
FORM CHARGER("SalesForm")

$Over:=Faux
Repeter
   $Total:=Imprimer objet(*;"malistbox")
   LISTBOX LIRE INFORMATION IMPRESSION(*;"malistbox";lk impression terminée;$Over)
   SAUT DE PAGE
Jusque($Over)

FERMER TACHE IMPRESSION
```

Exemple 2

Impression d'au moins 500 lignes de la list box, sachant que certaines lignes sont masquées :

```
$GlobalPrinted:=0
Repeter
   $Total:=Imprimer objet(*;"malistbox")
   LISTBOX LIRE INFORMATION IMPRESSION(*;"malistbox";lk nombre lignes imprimées;$Printed)
   $GlobalPrinted:=$GlobalPrinted+$Printed
   SAUT DE PAGE
Jusque($GlobalPrinted>=500)
```

LISTBOX Lire largeur colonne

LISTBOX Lire largeur colonne ({* ;} objet {; largeurMini {; largeurMaxi}}) -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
largeurMini	Entier long	⬅	Largeur minimale de la colonne (en pixels)
largeurMaxi	Entier long	⬅	Largeur maximale de la colonne (en pixels)
Résultat	Entier long	↩	Largeur de colonne en pixels

Description

La commande **LISTBOX Lire largeur colonne** retourne la largeur (en pixels) de la colonne de list box désignée par les paramètres *objet* et * . Vous pouvez passer indifféremment une colonne ou un en-tête de colonne de list box dans le paramètre *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

LISTBOX Lire largeur colonne peut retourner dans les paramètres *largeurMini* et *largeurMaxi* les limites de redimensionnement de la colonne. Ces limites peuvent être définies via la commande **LISTBOX FIXER LARGEUR COLONNE**. Si aucune valeur de largeur minimale et/ou maximale n'a été fixée pour la colonne, le paramètre correspondant retourne 0.

LISTBOX Lire nombre colonnes

LISTBOX Lire nombre colonnes ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	⇐	Nombre de colonnes

Description

La commande **LISTBOX Lire nombre colonnes** retourne le nombre total de colonnes (visibles ou non) présentes dans la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

LISTBOX Lire nombre lignes

LISTBOX Lire nombre lignes ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	⇐	Nombre de lignes

Description

La commande **LISTBOX Lire nombre lignes** retourne le nombre de lignes présentes dans la list box désignée par les paramètres *objet* et *.

Note : LISTBOX Lire nombre lignes ne tient pas compte du statut masqué/affiché des lignes. Par exemple, dans une list box contenant 10 lignes et dont les 9 premières sont masquées, **LISTBOX Lire nombre lignes** retournera 10.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Note : Si les tableaux associés aux colonnes d'un objet List box n'ont pas tous la même taille, seul le nombre d'éléments correspondant au plus petit tableau apparaît dans la list box et est retourné par cette commande.

LISTBOX LIRE OBJETS

LISTBOX LIRE OBJETS ({* ;} objet ; tabNomsObj)

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabNomsObj	Tableau texte	⬅	Noms des sous-objets de la list box (en-têtes, colonnes, pieds)

Description

La commande **LISTBOX LIRE OBJETS** retourne un tableau contenant les noms de chaque objet composant la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Passez dans *tabNomsObj* un tableau texte qui sera automatiquement rempli par la commande. Les noms des objets sont retournés dans leur ordre d'affichage, avec la séquence suivante :

nomCol1
nomEntêteCol1
nomPiedCol1
nomCol2
nomEntêteCol2
nomPiedCol2
...

Le tableau retourne les noms des objets de toutes les colonnes (y compris les pieds de colonnes), quel que soit leur statut visible/invisible.

Cette commande est utile dans le contexte de l'analyse d'un formulaire via les commandes **FORM CHARGER**, **FORM LIRE OBJETS** et **OBJET Lire type**. Elle permet, si nécessaire, d'obtenir les noms des sous-objets des list box.

Exemple

Vous souhaitez charger un formulaire et obtenir la liste de tous les objets des list box qu'il contient.

```
FORM CHARGER("MonFormulaire")
TABLEAU TEXTE(tabObjets;0)
FORM LIRE OBJETS(tabObjets)
TABLEAU ENTIER LONG(ar_type;Taille tableau(tabObjets))
Boucle($i;1;Taille tableau(tabObjets))
      ar_type{$i}:=OBJET Lire type(*;tabObjets{$i})
      Si(ar_type{$i}=Objet type listbox)
         TABLEAU TEXTE(tabObjetsLB;0)
         LISTBOX LIRE OBJETS(*;tabObjets{$i};tabObjetsLB)
      Fin de si
Fin de boucle
FORM LIBERER
```

LISTBOX LIRE POSITION CELLULE

LISTBOX LIRE POSITION CELLULE ({* ;} objet ; colonne ; ligne {; varCol})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
colonne	Entier long	⇐	Numéro de colonne
ligne	Entier long	⇐	Numéro de ligne
varCol	Pointeur	⇐	Pointeur sur la variable de colonne

Description

La commande **LISTBOX LIRE POSITION CELLULE** retourne les numéros de la *colonne* et de la *ligne* correspondant à l'emplacement du dernier clic ou de la dernière action de sélection effectuée dans la list box désignée par * et objet.
La commande retourne les coordonnées du clic ou de l'action de sélection même lorsque la saisie n'est pas autorisée dans la list box.

Notes :

- Le numéro retourné dans le paramètre *ligne* ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.
- Si le clic a eu lieu dans une cellule d'une fausse colonne, le paramètre *ligne* contient "*n*+1", où *n* est le nombre de colonnes de la list box (une fausse colonne peut être automatiquement ajoutée lorsque l'option "Redimensionnement colonnes auto" est sélectionnée ; pour plus d'informations, reportez-vous au paragraphe **Thème Redimensionnement**).

Si vous passez le paramètre facultatif *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable.

Le paramètre facultatif *varCol* retourne un pointeur sur la variable (c'est-à-dire le tableau) associée à la colonne.

Cette commande peut être appelée uniquement dans le cadre d'une list box générant l'un des événements formulaire suivants :

- Sur clic et Sur double clic
- Sur avant frappe clavier et Sur après frappe clavier
- Sur après modification
- Sur gain focus et Sur perte focus
- Sur données modifiées
- Sur nouvelle sélection
- Sur avant saisie

Lorsqu'elle est appelée en dehors de ce contexte, **LISTBOX LIRE POSITION CELLULE** retourne 0 dans *colonne* et *ligne*.

Cette commande tient compte des actions de sélection ou de désélection effectuées via la souris, les touches du clavier et la commande **EDITER ELEMENT** (qui génère l'événement Sur gain focus).

Si la sélection est modifiée via les touches fléchées du clavier, *colonne* retourne 0. Dans ce cas, s'il est passé, le paramètre *varCol* retourne **Nil**.

LISTBOX Lire style ligne

LISTBOX Lire style ligne ({* ;} objet ; ligne) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ligne	Entier long	⇒	Numéro de ligne
Résultat	Entier long	⇐	Valeur de style

Description

Note : Cette commande fonctionne uniquement avec les list box de type tableau.

La commande **LISTBOX Lire style ligne** retourne le style de police d'une ligne ou d'une cellule de la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Vous pouvez désigner comme paramètre *objet* une list box ou une colonne de list box :

- si *objet* désigne une list box, la commande retourne le style de la ligne
- si *objet* désigne une colonne, la commande retourne le style de la cellule

Passez dans *ligne* le numéro de la ligne dont vous souhaitez obtenir le style.

Note : La commande ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.

Attention, un style affecté à une ligne n'est pas forcément affiché dans toutes les cellules de ligne (cf. exemple). Si des valeurs de style contradictoires sont définies via les propriétés de la list box ou de la colonne, un ordre de priorité est appliqué. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Exemple

Soit la list box suivante :

text	text	text	text	text	text
text	text	text	text	text	text
text	*text*	*text*	*text*	**text**	*text*
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
vStyl:=LISTBOX Lire style ligne(*;"Col5";3)
vStyl2:=LISTBOX Lire style ligne(*;"List Box";3)
    // vStyl contient 1 (Gras)
    // vStyl2 contient 6 (Italique + Souligné)
```

LISTBOX LIRE TABLE SOURCE

LISTBOX LIRE TABLE SOURCE ({* ;} objet ; numTable {; nom {; nomSurlignage}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
numTable	Entier long	⇐	Numéro de la table de la sélection
nom	Chaîne	⇐	Nom de la sélection temporaire ou "" pour la sélection courante
nomSurlignage	Chaîne	⇐	Nom de l'ensemble de surlignage

Description

La commande **LISTBOX LIRE TABLE SOURCE** permet de connaître la source courante des données affichées dans la list box désignée par les paramètres * et *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

La commande retourne dans le paramètre *numTable* le numéro de la table principale associée à la list box et dans le paramètre facultatif *nom* le nom de la sélection temporaire éventuellement utilisée.
Si les lignes de la list box sont liées à la sélection courante de la table, le paramètre *nom*, s'il est passé, retourne une chaîne vide. Si les lignes de la list box sont liées à une sélection temporaire, le paramètre *nom* retourne le nom de cette sélection temporaire.

Si la list box est associée à des tableaux, *numTable* retourne -1 et *nom*, s'il est passé, retourne une chaîne vide.

LISTBOX Lire tableau

LISTBOX Lire tableau ({* ;} objet ; typeTab) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
typeTab	Entier long	⇒	Type de tableau
Résultat	Pointeur	⇐	Pointeur vers le tableau associé à la propriété

Description

Note : Cette commande fonctionne uniquement avec les list box de type tableau.

La commande **LISTBOX Lire tableau** retourne un pointeur vers le tableau *typeTab* de la list box ou de la colonne de list box désignée par les paramètres *objet* et *.

Des tableaux de style, de couleur, de couleur de fond ou de contrôle des lignes peuvent être associés aux list box de type tableau ou (hormis le tableau de contrôle des lignes) aux colonnes de list box tableau via la Liste des propriétés en mode Développement ou la commande **LISTBOX FIXER TABLEAU**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Vous pouvez désigner comme paramètre *objet* une list box ou une colonne de list box.

Passez dans *typeTab* le type du tableau de propriété à obtenir. Vous pouvez utiliser une des constantes suivantes du thème "**List box**" :

Constante	Type	Valeur	Comment
lk tableau contrôle	Entier long	3	
lk tableau couleur de fond	Entier long	1	
lk tableau couleur de police	Entier long	0	
lk tableau hauteurs lignes	Entier long	4	(Licence 4D View Pro requise)
lk tableau style	Entier long	2	

La commande retourne une des valeurs suivantes :

- **Nil** si aucun tableau de la propriété demandée n'est associé à la colonne ou à la list box
- un pointeur vers le tableau de la propriété demandée, défini par l'utilisateur
- un pointeur vers le tableau de la propriété demandée, défini dynamiquement lors de l'appel de la commande **LISTBOX FIXER COULEUR LIGNE** ou **LISTBOX FIXER STYLE LIGNE**.

Exemple

Exemples type d'utilisation :

```
vPtr:=LISTBOX Lire tableau(*;"MyLB";lk tableau couleur de police)
    // retourne un pointeur vers le tableau de couleurs de police associé
    // à la list box "MyLB"

vPtr:=LISTBOX Lire tableau(*;"Col4";lk tableau style)
    // retourne un pointeur vers le tableau de styles de police associé
    // à la colonne de list box "Col4"
```

LISTBOX LIRE TABLEAUX

LISTBOX LIRE TABLEAUX ({* ;} objet ; tabNomsCols ; tabNomsEntêtes ; tabVarCols ; tabVarEntêtes ; tabColsVisibles ; tabStyles {; tabNomsPieds ; tabVarPieds})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabNomsCols	Tableau chaîne	⇐	Noms d'objet des colonnes
tabNomsEntêtes	Tableau chaîne	⇐	Noms d'objet des en-têtes
tabVarCols	Tableau pointeur	⇐	Pointeurs vers les variables des colonnes ou Pointeurs vers les champs des colonnes ou Nil
tabVarEntêtes	Tableau pointeur	⇐	Pointeurs vers les variables des en-têtes
tabColsVisibles	Tableau booléen	⇐	Visibilité de chaque colonne
tabStyles	Tableau pointeur	⇐	Pointeurs vers les tableaux ou les variables de styles de couleurs et de contrôle des lignes ou Nil
tabNomsPieds	Tableau chaîne	⇐	Noms d'objet des pieds de colonnes
tabVarPieds	Tableau pointeur	⇐	Pointeurs vers les variables des pieds de colonnes

Description

La commande **LISTBOX LIRE TABLEAUX** retourne un ensemble de tableaux synchronisés fournissant diverses informations sur chaque colonne (visible ou non) de la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

A l'issue de l'exécution de la commande :

- Le tableau *tabNomsCols* contient la liste des noms d'objet de chaque colonne de la list box.
- Le tableau *tabNomsEntêtes* contient la liste des noms d'objet de chaque en-tête de colonne de la list box.
- Le tableau *tabVarCols* contient, pour une list box de type tableau, des pointeurs vers les variables (c'est-à-dire les tableaux) associées à chaque colonne de la list box. Pour une list box de type sélection, *tabVarCols* contient :
  - pour une colonne associée à un champ, un pointeur vers le champ associé,
  - pour une colonne associée à une variable, un pointeur vers la variable,
  - pour une colonne associée à une expression, un pointeur Nil.
- Le tableau *tabVarEntêtes* contient des pointeurs vers les variables associées à chaque en-tête de colonne de la list box.
- Le tableau *tabColsVisibles* contient une valeur booléenne pour chaque colonne, indiquant si la colonne est visible (valeur **Vrai**) ou masquée (valeur **Faux**) dans la list box.
- Le tableau *tabStyles* contient, pour une list box de type tableau, quatre pointeurs vers les quatre tableaux permettant d'appliquer individuellement un style, une couleur de police, une couleur de fond et un contrôle d'affichage personnalisés à chaque ligne de la list box. Ces tableaux sont associés à la list box dans la Liste des propriétés en mode Développement ou via la commande **LISTBOX FIXER TABLEAU**. Si un tableau n'est pas spécifié pour la list box, l'élément correspondant de *tabStyles* contient un pointeur Nil.
  Le quatrième pointeur correspond soit à un tableau booléen (tableau de lignes masquées), soit à un tableau d'entiers longs (tableau permettant de définir les lignes masquées, désactivées et non sélectionnables) en fonction de l'implémentation utilisée pour le tableau de contrôle des lignes (cf. section **Propriétés spécifiques des List box**).
  Pour une list box de type sélection, *tabStyles* contient :
  - pour chaque paramétrage défini via une variable, un pointeur vers la variable,
  - pour chaque paramétrage défini via une expression, un pointeur Nil.

LISTBOX NUMERO COLONNE DEPLACEE

LISTBOX NUMERO COLONNE DEPLACEE ({* ;} objet ; ancPosition ; nouvPosition)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ancPosition	Entier long	⇐	Ancienne position de la colonne déplacée
nouvPosition	Entier long	⇐	Nouvelle position de la colonne déplacée

Description

La commande **LISTBOX NUMERO COLONNE DEPLACEE** retourne dans les paramètres *ancPosition* et *nouvPosition* des numéros indiquant respectivement la précédente position et la nouvelle position de la colonne déplacée dans la list box désignée par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Cette commande doit être utilisée en combinaison avec l'événement formulaire Sur déplacement colonne (cf. commande **Evenement formulaire**).

Note : Cette commande tient compte des colonnes invisibles.

LISTBOX NUMERO LIGNE DEPLACEE

LISTBOX NUMERO LIGNE DEPLACEE ({* ;} objet ; ancPosition ; nouvPosition)

Paramètre	Type		Description
*	Opérateur	⇨	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇨	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ancPosition	Entier long	⇦	Précédente position de la ligne déplacée
nouvPosition	Entier long	⇦	Nouvelle position de la ligne déplacée

Description

La commande **LISTBOX NUMERO LIGNE DEPLACEE** retourne dans les paramètres *ancPosition* et *nouvPosition* des numéros indiquant respectivement la précédente position et la nouvelle position de la ligne déplacée dans la list box désignée par les paramètres *objet* et *.

Note : Le déplacement de lignes n'est possible que dans les list box de type tableau.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Cette commande doit être utilisée en combinaison avec l'événement formulaire Sur déplacement colonne (cf. commande **Evenement formulaire**).

Note : Cette commande ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.

LISTBOX SELECTIONNER LIGNE

LISTBOX SELECTIONNER LIGNE ({* ;} objet ; positionLigne {; action})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionLigne	Entier long	⇒	Numéro de la ligne à sélectionner
action	Entier long	⇒	Action de sélection

Description

La commande **LISTBOX SELECTIONNER LIGNE** provoque la sélection de la ligne de numéro *positionLigne* dans l'objet list box désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Le paramètre *action*, s'il est passé, permet de définir l'action de sélection à effectuer lorsqu'une sélection de lignes existe déjà dans la list box. Vous pouvez passer une valeur ou l'une des constantes suivantes, placées dans le thème **List box** :

Constante	Type	Valeur	Comment
lk ajouter à sélection	Entier long	1	La ligne sélectionnée est ajoutée à la sélection existante. Si la ligne désignée appartient déjà à la sélection existante, la commande ne fait rien.
lk remplacer sélection	Entier long	0	La ligne sélectionnée devient la nouvelle sélection et remplace la sélection existante. La commande produit le même effet qu'un clic de l'utilisateur sur une ligne de la list box (l'événement Sur clic n'est toutefois pas généré). Cette action est effectuée par défaut (lorsque le paramètre *action* n'est pas passé).
lk supprimer de sélection	Entier long	2	La ligne sélectionnée est supprimée de la sélection existante. Si la ligne désignée n'appartient pas à la sélection existante, la commande ne fait rien.

Lorsque le paramètre *positionLigne* ne correspond pas strictement à un numéro de ligne existante, la commande agit de la manière suivante :

- Si *positionLigne* est <0, la commande ne fait rien, quelle que soit la valeur du paramètre *action*.
- Si *positionLigne* vaut 0 et si le paramètre *action* contient lk remplacer sélection ou est omis, toutes les lignes de la listbox sont sélectionnées. Si le paramètre *action* contient lk supprimer de sélection, toutes les lignes de la listbox sont désélectionnées.
- Si la valeur de *positionLigne* est supérieure au nombre total de lignes contenues dans la listbox (dans le cas d'une listbox de type tableau uniquement), le tableau booléen associé à la listbox est automatiquement redimensionné et l'action de sélection est effectuée. Ce mécanisme permet d'utiliser **LISTBOX SELECTIONNER LIGNE** avec des commandes "standard" de gestion de tableaux (telles que **AJOUTER A TABLEAU**) n'entraînant pas de synchronisation immédiate de la listbox.
  A l'issue de l'exécution de la méthode, les tableaux sont synchronisés : si le tableau source de la listbox a effectivement été redimensionné, l'action de sélection est effectuée. Sinon, le tableau booléen associé à la listbox reprend sa taille initiale et la commande ne fait rien.

Notes :

- Si vous souhaitez que la list box défile de manière à afficher la ligne nouvellement sélectionnée, utilisez la commande **OBJET FIXER DEFILEMENT**.
- Pour passer une ligne en mode édition (saisie), utilisez la commande **EDITER ELEMENT**.
- Si le numéro passé dans *positionLigne* correspond à une ligne masquée dans la list box, la ligne est sélectionnée mais n'est pas affichée.

LISTBOX SELECTIONNER RUPTURE

LISTBOX SELECTIONNER RUPTURE ({* ;} objet ; ligne ; colonne {; action})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Variable (si * omis)
ligne	Entier long	⇒	Numéro de ligne de la rupture
colonne	Entier long	⇒	Numéro de colonne de la rupture
action	Entier long	⇒	Action de sélection

Description

La commande **LISTBOX SELECTIONNER RUPTURE** permet de sélectionner des lignes de rupture dans l'objet list box désigné par les paramètres *objet* et *. La list box doit être affichée en mode hiérarchique.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Les lignes de rupture sont ajoutées pour représenter la hiérarchie mais ne correspondent pas à des lignes de tableaux existantes. Pour désigner une ligne de rupture à sélectionner, vous devez passer dans les paramètres *ligne* et *colonne* des numéros de ligne et de colonne correspondant à la première occurrence dans le tableau correspondant. Ces valeurs sont retournées par la commande **LISTBOX LIRE POSITION CELLULE** lorsque l'utilisateur a sélectionné une ligne de rupture. Ce principe est détaillé dans le paragraphe "Gestion des lignes de rupture" de la section **Gestion des List box hiérarchiques**.

Le paramètre *action*, s'il est passé, permet de définir l'action de sélection à effectuer lorsqu'une sélection de lignes de rupture existe déjà dans la list box. Vous pouvez passer une valeur ou l'une des constantes suivantes, placées dans le thème "**List box**" :

Constante	Type	Valeur	Comment
lk ajouter à sélection	Entier long	1	La ligne sélectionnée est ajoutée à la sélection existante. Si la ligne désignée appartient déjà à la sélection existante, la commande ne fait rien.
lk remplacer sélection	Entier long	0	La ligne sélectionnée devient la nouvelle sélection et remplace la sélection existante. La commande produit le même effet qu'un clic de l'utilisateur sur une ligne de la list box (l'événement Sur clic n'est toutefois pas généré). Cette action est effectuée par défaut (lorsque le paramètre *action* n'est pas passé).
lk supprimer de sélection	Entier long	2	La ligne sélectionnée est supprimée de la sélection existante. Si la ligne désignée n'appartient pas à la sélection existante, la commande ne fait rien.

Exemple

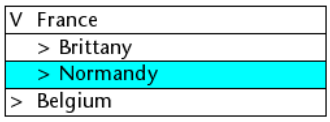
Soient les tableaux suivants représentés dans une list box :

(T1)	(T2)	(T3)	(T4)
France	Brittany	Brest	120000
France	Brittany	Quimper	80000
France	Brittany	Rennes	200000
France	Normandy	Caen	220000
France	Normandy	Deauville	4000
France	Normandy	Cherbourg	41000
Belgium	Wallonia	Namur	111000
Belgium	Wallonia	Liège	200000
Belgium	Flanders	Antwerp	472000
Belgium	Flanders	Louvain	95000

Nous souhaitons sélectionner la ligne de rupture "Normandie" dans la représentation hiérarchique de ces tableaux :

```
$ligne:=Chercher dans tableau(T2;"Normandie")
$colonne:=2
LISTBOX CONTRACTER(*;"MaListbox") `contraction de tous les niveaux
LISTBOX SELECTIONNER RUPTURE(*;"MaListbox";$ligne;$colonne)
```

Voici le résultat :



LISTBOX SUPPRIMER COLONNE

LISTBOX SUPPRIMER COLONNE ({* ;} objet ; positionCol {; nombre})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionCol	Entier long	⇒	Numéro courant de la colonne à supprimer
nombre	Entier long	⇒	Nombre de colonnes à supprimer

Description

La commande **LISTBOX SUPPRIMER COLONNE** supprime une ou plusieurs colonne(s) (visibles ou non) dans la list box désignée par les paramètres *objet* et *.

Note : Cette commande ne fait rien si elle est appliquée à la première colonne d'une list box affichée en mode hiérarchique.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Si vous ne passez pas le paramètre facultatif *nombre*, la commande supprime simplement la colonne désignée par le paramètre *positionCol*.
Sinon, le paramètre *nombre* indique le nombre de colonnes à supprimer vers la droite à partir de la colonne *positionCol* (celle-ci incluse).

Si le paramètre *positionCol* est supérieur au nombre de colonnes de la list box, la commande ne fait rien.

LISTBOX SUPPRIMER LIGNES

LISTBOX SUPPRIMER LIGNES ({* ;} objet ; positionLigne {; nbLignes})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionLigne	Entier long	⇒	Numéro de la première ligne à supprimer
nbLignes	Entier long	⇒	Nombre de lignes à supprimer

Description

La commande **LISTBOX SUPPRIMER LIGNES** supprime une ou plusieurs ligne(s) à partir de la ligne numéro *positionLigne* (visible ou non) de la list box désignée par les paramètres *objet* et *.

Note : Cette commande fonctionne uniquement avec les list box basées sur des tableaux. Lorsque cette commande est utilisée avec une list box basée sur des sélections, elle ne fait rien et la variable système OK retourne 0.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

La ligne *positionLigne* est supprimée automatiquement de tous les tableaux composant la list box.
Notez qu'après l'exécution de la commande, il n'y a plus d'élément sélectionné dans la list box.

Si le paramètre *positionLigne* est supérieur au nombre de lignes des tableaux de la list box ou s'il est inférieur à 1, la commande ne fait rien.

Note : Cette commande ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.

LISTBOX TRIER COLONNES

LISTBOX TRIER COLONNES ({* ;} objet ; numColonne ; sensDuTri {; numColonne2 ; sensDuTri2 ; ... ; numColonneN ; sensDuTriN})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
numColonne	Entier long	⇒	Numéro(s) de colonne(s) de tri
sensDuTri	Opérateur	⇒	">"pour effectuer un tri croissant ou "<" pour effectuer un tri décroissant

Description

La commande **LISTBOX TRIER COLONNES** permet de trier (réordonner) toutes les lignes de la list box désignée par les paramètres *objet* et * sur la base des valeurs d'une ou plusieurs colonne(s).

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Passez dans *numColonne* le numéro de la colonne dont les valeurs seront utilisées comme critère de tri. Vous pouvez utiliser tout type de données, à l'exception des colonnes contenant des images et des pointeurs.

Passez dans *sensDuTri* le symbole > ou < indiquant l'ordre du tri : croissant ou décroissant. Si *sensDuTri* est égal au symbole "supérieur à" (>), l'ordre du tri est croissant. Si *sensDuTri* est égal au symbole "inférieur à" (<), l'ordre du tri est décroissant.

Vous pouvez définir des tris multi-niveaux : pour cela, passez autant de paires *numColonne;sensDuTri* que nécessaire. Le niveau de tri est défini par la position du paramètre lors de l'appel.

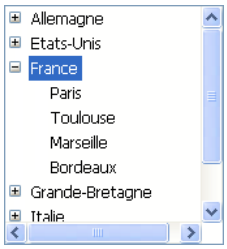
Conformément au principe de fonctionnement des list box, les colonnes sont synchronisées, ce qui signifie que le tri d'une colonne est automatiquement répercuté sur toutes les colonnes de l'objet.

Listes hiérarchiques

- Gestion des listes hiérarchiques
- AJOUTER A LISTE
- CHANGER ELEMENT
- CHANGER PROPRIETES ELEMENT
- CHANGER PROPRIETES LISTE
- Charger liste
- Chercher dans liste
- Copier liste
- Element parent
- Elements selectionnes
- FIXER ICONE ELEMENT
- FIXER PARAMETRE ELEMENT
- FIXER POLICE ELEMENT
- INFORMATION ELEMENT
- INSERER DANS LISTE
- LIRE ICONE ELEMENT
- LIRE PARAMETRE ELEMENT
- LIRE PARAMETRE ELEMENT TABLEAUX
- Lire police element
- LIRE PROPRIETES ELEMENT
- LIRE PROPRIETES LISTE
- LISTE DES LISTES
- Liste existante
- Nombre elements
- Nouvelle liste
- Position element liste
- SELECTIONNER ELEMENTS PAR POSITION
- SELECTIONNER ELEMENTS PAR REFERENCE
- STOCKER LISTE
- SUPPRIMER DANS LISTE
- SUPPRIMER LISTE
- TRIER LISTE
- *_o_REDESSINER LISTE*

Gestion des listes hiérarchiques

Les listes hiérarchiques sont des objets de formulaire permettant d'afficher des données sous forme de listes comportant un ou plusieurs niveaux qu'il est possible de déployer ou de contracter.



Dans les formulaires, les listes hiérarchiques peuvent servir à l'affichage ou la saisie de données. Chaque élément de liste peut contenir jusqu'à de 2 milliards de caractères (taille maximale d'un champ texte) et être associé à une icône. Les listes hiérarchiques prennent généralement en charge les clics, double-clics, navigation au clavier ou encore le glisser-déposer. Il est possible d'effectuer une recherche parmi le contenu d'une liste (commande **Chercher dans liste**).

Création et modification

Les listes hiérarchiques peuvent être créées entièrement par programmation (via les commandes **Nouvelle liste** ou **Copier liste**) ou à partir d'énumérations définies dans l'éditeur d'énumérations en mode Développement (commande **Charger liste**).

Le contenu et l'apparence des listes hiérarchiques sont gérés par programmation, à l'aide des commandes du thème "Liste hiérarchique". Certaines caractéristiques d'apparence spécifiques peuvent également être définies via les commandes génériques du thème **Objets de formulaires** (cf. ci-dessous).

Vous pouvez associer dynamiquement des références de listes hiérarchiques aux énumérations des objets de formulaires (sources, valeurs obligatoires et valeurs exclues) à l'aide des commandes **OBJET FIXER LISTE PAR REFERENCE** ou **OBJET FIXER LISTE PAR NOM**. Vous pouvez également associer des énumérations définies dans l'éditeur d'énumérations aux objets de formulaires via la Liste des propriétés.

RefListe et nom d'objet

Une liste hiérarchique est à la fois un **objet de langage** existant en mémoire et un **objet de formulaire**.
L'**objet de langage** est référencé de manière unique par un identifiant interne, de type Entier long, désigné par *RefListe* dans ce manuel. Cet identifiant est retourné par les commandes permettant de créer des listes **Nouvelle liste**, **Copier liste**, **Charger liste**, **BLOB vers liste**. Il n'existe qu'une seule instance en mémoire de l'objet de langage et toute modification effectuée sur cet objet est immédiatement répercutée dans tous les endroits où il est utilisé.
L'**objet de formulaire** n'est pas nécessairement unique : il peut exister plusieurs représentations d'une même liste hiérarchique dans un même formulaire ou dans des formulaires différents. Comme pour les autres objets de formulaire, vous désignez l'objet dans le langage via la syntaxe (*;"NomListe"...).

Vous connectez l'"objet de langage" liste hiérarchique avec l'"objet de formulaire" liste hiérarchique par l'intermédiaire de la variable contenant la valeur de l'identifiant unique *RefListe*. Par exemple, si vous écrivez :

```
maliste:=Nouvelle liste
```

... il suffit d'associer le nom de variable maliste à l'objet de formulaire Liste hiérarchique dans la liste des propriétés afin qu'il gère l'objet de langage dont la *RefListe* est stockée dans maListe.

Chaque représentation de liste dispose de caractéristiques propres et partage des caractéristiques communes avec l'ensemble des représentations. Les caractéristiques propres à chaque représentation de liste sont les suivantes :

- la sélection,
- l'état déployé/contracté des éléments,
- la position du curseur de défilement.
  Les autres caractéristiques (police, style, filtre de saisie, couleur, contenu de la liste, icônes, etc.) sont communes à toutes les représentations et ne peuvent pas être modifiées séparément.
  Par conséquent, lorsque vous utilisez des commandes se basant sur la configuration déployé/contracté ou l'élément courant, par exemple **Nombre elements** (lorsque le paramètre * final n'est pas passé), il importe de pouvoir désigner sans ambiguïté la représentation à utiliser.

Vous devez utiliser l'identifiant de type *RefListe* avec les commandes du langage lorsque vous souhaitez désigner la liste hiérarchique résidant en mémoire. Si vous souhaitez désigner la représentation au niveau du formulaire d'un objet Liste hiérarchique, vous devez utiliser le nom de l'objet (type chaîne) dans la commande, via la syntaxe (*;"NomListe"...). Cette syntaxe est identique à celle en vigueur dans les commandes du thème "Propriétés des objets". Elle est acceptée par la plupart des commandes du thème "Liste hiérarchique" agissant sur les propriétés des listes (reportez-vous à la description des commandes du thème).

Attention, dans le cas des commandes définissant des propriétés, la syntaxe basée sur le nom d'objet ne signifie pas que seul l'objet de formulaire désigné sera modifié par la commande, mais que l'action de la commande sera basée sur l'état de cet objet. Les caractérisques communes des listes hiérarchiques sont toujours modifiées dans toutes les représentations.
Par exemple, si vous passez l'instruction **FIXER POLICE ELEMENT(*;"maliste1";*;lapolice)**, vous indiquez que vous souhaitez modifier la police d'un élément de la liste hiérarchique associée à l'objet de formulaire maliste1. La commande tiendra compte de l'élément courant de l'objet maliste1 pour définir l'élément à modifier, mais cette modification sera reportée dans toutes les représentations de la liste dans tous les process.

Prise en compte du @

Comme pour les autres commandes de gestion des propriété d'objets, il est possible d'utiliser le caractère "@" dans le paramètre **NomListe**. En principe, cette syntaxe permet de désigner un ensemble d'objets dans le formulaire. Toutefois, dans le contexte des commandes de liste hiérarchique, ce principe n'est pas applicable dans tous les cas. Cette syntaxe aura deux effets différents en fonction du type de commande :

- Pour les commandes fixant des propriétés, cette syntaxe désigne tous les objets dont le nom correspond (fonctionnement standard). Par exemple, le

paramètre "LH@" désigne tous les objets de type liste hiérarchique dont le nom débute par “LH”. Ces commandes sont :
SUPPRIMER DANS LISTE,
INSERER DANS LISTE
SELECTIONNER ELEMENTS PAR POSITION
CHANGER ELEMENT
FIXER POLICE ELEMENT
FIXER ICONE ELEMENT
FIXER PARAMETRE ELEMENT
CHANGER PROPRIETES ELEMENT

- Pour les commandes récupérant des propriétés, cette syntaxe désigne le premier objet dont le nom correspond. Ces commandes sont :
Nombre elements
Chercher dans liste
INFORMATION ELEMENT
Lire police element
LIRE ICONE ELEMENT
LIRE PARAMETRE ELEMENT
LIRE PROPRIETES ELEMENT
Element parent
Position element liste
Elements selectionnes

Commandes génériques utilisables avec les listes hiérarchiques

Il est possible de modifier l’apparence d’une liste hiérarchique dans un formulaire à l’aide de plusieurs commandes 4D génériques. Vous devez passer à ces commandes soit le nom d’objet de la liste hiérarchique (en utilisant le paramètre *), soit son nom de variable (syntaxe standard).

Note : Dans le cas des listes hiérarchiques, la variable de formulaire contient la valeur de *RéfListe*. Si vous exécutez une commande de modification d’attribut en lui passant la variable associée à la liste hiérarchique, il ne sera pas possible de définir la liste cible en cas de multi-représentation. Seul le nom d’objet permet donc de différencier individuellement chaque représentation.

Voici la liste des commandes utilisables avec les listes hiérarchiques :
OBJET FIXER POLICE
OBJET FIXER STYLE POLICE
OBJET FIXER TAILLE POLICE
OBJET FIXER COULEUR
OBJET FIXER FILTRE SAISIE
OBJET FIXER SAISISSABLE
OBJET FIXER BARRES DEFILEMENT
OBJET FIXER DEFILEMENT
OBJET FIXER COULEURS RVB

Rappel : A l'exception de la commande **OBJET FIXER DEFILEMENT**, ces commandes modifient toutes les représentations d'une même liste, même si vous désignez une liste via son nom d'objet.

Priorité des commandes de propriété

Certaines propriétés d’une liste hiérarchique (par exemple l’attribut saisissable ou la couleur) peuvent être définies de trois manières : via la Liste des propriétés en mode Développement, via une commande du thème “Propriétés des objets” ou via une commande du thème “Liste hiérarchique”.
Lorsque ces trois moyens sont utilisés pour définir les propriétés d’une liste, l’ordre de priorité suivant est appliqué :
1. Commandes du thème “Liste hiérarchique”
2. Commandes générique de propriété d'objet
3. Paramètres de la Liste des propriétés

Ce principe est appliqué quel que soit l’ordre d’appel des commandes. Si une propriété d’élément est modifiée individuellement via une commande de liste hiérarchique, la commande de propriété d’objet équivalente sera sans effet sur cet élément même si elle est appelée ultérieurement. Par exemple, si vous modifiez la couleur d’un élément via la commande **CHANGER PROPRIETES ELEMENT**, la commande **OBJET FIXER COULEUR** n’aura aucun effet sur cet élément.

Gestion des éléments par position ou par référence

Vous pouvez généralement travailler de deux manières avec le contenu des listes hiérarchiques : par position ou par référence.

- Lorsque vous travaillez par position, 4D se base sur la position relative des éléments dans la liste affichée à l'écran pour les identifier. Le résultat sera différent selon que certains éléments hiérarchiques sont déployés ou non. A noter qu'en cas de multi-représentation, chaque objet de formulaire comporte sa propre configuration d'éléments contractés/déployés.
- Lorsque vous travaillez par référence, 4D se base sur le numéro unique *réfElément* des éléments de la liste. Chaque élément peut être ainsi désigné, quelle que soit sa position ou son affichage dans la liste hiérarchique.

Exploiter les numéros de référence des éléments (réfElément)

Chaque élément d'une liste hiérarchique dispose d'un numéro de référence (*réfElément*) de type Entier long. Cette valeur est destinée uniquement à votre propre usage : 4D ne fait que la maintenir.

Attention : Vous pouvez utiliser comme numéro de référence toute valeur de type entier long, sauf la valeur 0. En effet, pour la plupart des commandes de ce thème, la valeur 0 permet de désigner le dernier élément ajouté à la liste.

Voici quelques astuces quant à l'utilisation du numéro de référence unique :

(1) Vous n'avez pas besoin d'identifier chaque élément de façon unique (niveau débutant).

- Premier exemple : vous construisez par programmation un système d'onglets, par exemple, un carnet d'adresses. Comme le système vous retournera le numéro de l'onglet sélectionné, vous n'aurez probablement pas besoin de davantage d'informations. Dans ce cas, ne vous préoccupez pas des numéros de référence des éléments : passez n'importe quelle valeur (hormis 0) dans le paramètre *réfElément*. Notez que pour un système de carnet d'adresses, vous pouvez prédéfinir une liste A, B,..., Z en mode Développement. Vous pouvez également la créer par programmation afin d'éliminer les lettres pour lesquelles il n'y a pas d'enregistrement.
- Deuxième exemple : en travaillant avec une base, vous construisez progressivement une liste de mots-clés. Vous pouvez sauvegarder la liste à la fin de chaque session, en utilisant les commandes **STOCKER LISTE** ou **LISTE VERS BLOB**, et la recharger au début de chaque session, à l'aide des

commandes **Charger liste** ou **BLOB vers liste**. Vous pouvez afficher cette liste dans une palette flottante ; lorsque l'utilisateur clique sur un mot-clé de la liste, l'élément choisi est inséré dans la zone saisissable sélectionnée du process de premier plan. Vous pouvez également utiliser le glisser-déposer. En tout état de cause, l'important est que vous ne traitez que l'élément sélectionné (par clic ou glisser-déposer), car les commandes **Elements selectionnes** (en cas de clic) et **PROPRIETES GLISSER DEPOSER** vous retournent la position de l'élément que vous devez traiter. En utilisant cette valeur de position, vous obtenez le libellé de l'élément grâce à la commande **INFORMATION ELEMENT**. Ici aussi, vous n'avez pas besoin d'identifier de façon unique chaque élément ; vous pouvez passer n'importe quelle valeur (hormis 0) dans le paramètre *réfElément*.

(2) Vous avez besoin d'identifier partiellement les éléments de la liste (niveau intermédiaire).
Vous utilisez le numéro de référence de l'élément pour stocker l'information nécessaire lorsque vous devez agir sur un élément ; ce point est détaillé dans l'exemple de la commande **AJOUTER A LISTE**. Dans cet exemple, nous utilisons les numéros de référence des éléments pour stocker des numéros d'enregistrements. Cependant, nous devons pouvoir établir une distinction entre les éléments qui correspondent aux enregistrements [Départements] et ceux qui correspondent aux enregistrements [Employés].

(3) Vous avez besoin d'identifier les éléments de la liste de façon unique (niveau avancé).
Vous programmez une gestion élaborée de listes hiérarchiques, dans laquelle vous devez absolument pouvoir identifier chaque élément de manière unique à tous les niveaux de la liste. Un moyen simple d'implémenter ce fonctionnement est de maintenir un compteur personnel. Supposons que vous créez une liste hlList à l'aide de la commande **AJOUTER A LISTE**. A ce stade, vous initialisez un compteur vlhCounter à 1. A chaque fois que vous appelez **AJOUTER A LISTE** ou **INSERER DANS LISTE**, vous incrémentez ce compteur (*vlhCounter:=vlhCounter+1*), et vous passez le compteur comme numéro de référence de l'élément. L'astuce consiste à ne pas décrémenter le compteur lorsque vous détruisez des éléments — le compteur ne peut qu'augmenter. En procédant ainsi, vous garantissez l'unicité des numéros de référence des éléments. Puisque ces numéros sont des valeurs de type Entier long, vous pouvez ajouter ou insérer plus de deux milliards d'éléments dans une liste qui a été réinitialisée... (si vous manipulez d'aussi grandes quantités d'éléments, cela signifie généralement que vous devriez utiliser une table plutôt qu'une liste.)

Note : Si vous exploitez les **Opérateurs sur les bits**, vous pouvez également utiliser les numéros de référence des éléments pour stocker des informations qui peuvent être logées dans un Entier long, c'est-à-dire 2 Entiers, des valeurs de 4 octets ou encore 32 Booléens.

Quand avez-vous besoin de numéros de référence uniques ?

Dans la plupart des cas, lorsque vous utilisez des listes hiérarchiques pour des besoins d'interface utilisateur, pour lesquels seul l'élément sélectionné (par un clic ou par glisser-déposer) est important, vous n'avez pas besoin d'utiliser les numéros de référence des éléments. Les commandes **Elements selectionnes** et **INFORMATION ELEMENT** vous fournissent toutes les informations nécessaires à la gestion de l'élément sélectionné. De plus, des commandes telles que **INSERER DANS LISTE** et **SUPPRIMER DANS LISTE** vous permettent de manipuler la liste de manière "relative" à l'élément sélectionné.
En pratique, vous devez vous préoccuper des numéros de référence d'éléments lorsque vous voulez accéder directement par programmation à n'importe quel élément de la liste, et pas nécessairement à l'élément couramment sélectionné.

AJOUTER A LISTE

AJOUTER A LISTE (liste ; libelléElément ; réfElément {; sous_Liste ; déployée})

Paramètre	Type		Description
liste	RefListe	⇒	Numéro de référence de liste
libelléElément	Chaîne	⇒	Libellé du nouvel élément
réfElément	Entier long	⇒	Numéro de référence unique du nouvel élément
sous_Liste	RefListe	⇒	Sous-liste optionnelle à rattacher au nouvel élément
déployée	Booléen	⇒	Indique si la sous-liste doit être déployée ou non

Description

La commande **AJOUTER A LISTE** ajoute un nouvel élément à la liste hiérarchique dont vous avez passé le numéro de référence dans le paramètre *liste*.

Vous passez le libellé de l'élément dans le paramètre *libelléElément*. Vous pouvez passer une expression de type Alpha ou Texte, pouvant contenir jusqu'à 2 milliards de caractères.

Vous passez le numéro de référence unique de l'élément (de type Entier long) dans le paramètre *réfElément*. Même si nous qualifions ce numéro de référence d'élément comme unique, vous pouvez en réalité passer la valeur que vous voulez. Reportez-vous à la section **Gestion des listes hiérarchiques** pour plus d'informations sur le paramètre *réfElément*.

Si vous souhaitez également que l'élément comporte des sous-éléments, passez un numéro de référence de liste valide dans le paramètre *sous_Liste*. Dans ce cas, vous devez également passer le paramètre *déployée*. Passez **Vrai** ou **Faux** dans ce paramètre pour que cette sous-liste s'affiche respectivement déployée ou contractée.

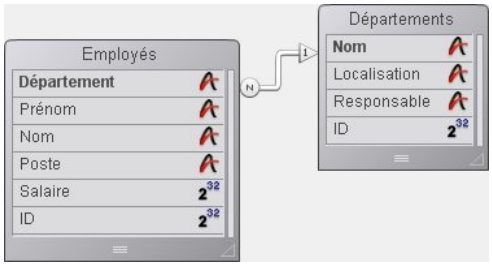
La référence de la liste que vous passez dans *sous_Liste* doit être une liste existante. Elle peut comporter un seul niveau ou contenir elle-même des sous-listes. Si vous ne voulez pas rattacher de sous-liste au nouvel élément, omettez le paramètre ou passez 0. Si vous passez le paramètre *sous_Liste* et ne passez pas le paramètre *déployée*, la sous-liste apparaît par défaut contractée.

Conseils :

- Pour insérer un nouvel élément dans une liste, utilisez **INSERER DANS LISTE**. Pour modifier le libellé d'un élément existant ou sa sous-liste, ainsi que son état déployé/contracté, utilisez **CHANGER ELEMENT**.
- Pour changer l'apparence de l'élément ajouté, utilisez **CHANGER PROPRIETES ELEMENT**.

Exemple

Voici une vue partielle de la structure d'une base :



Les tables [Départements] et [Employés] contiennent les enregistrements suivants :

Nom :	Localisation :	Responsable :
Biologie marine	Vivariurm 11	Robert Ibéri
Comptabilité	2e étage	Louis Berouet
Ventes	RDC Ouest	Eliane Bergis

Département :	Prénom :	Nom :
Biologie marine	Daphné	Vaudelles
Comptabilité	Gérard	Périet
Ventes	Guy	Chartret
Comptabilité	Gilbert	Darieux
Biologie marine	Arnaud	Leterrier
Ventes	Frédérique	Dalhoum
Biologie marine	Pierre	Adamo

Vous voulez utiliser une liste hiérarchique, appelée *hlList*, qui affiche les départements, et pour chaque département, une sous-liste contenant les employés travaillant dans ce département. La méthode objet de *hlList* est la suivante:

```
   ` Méthode objet Liste hiérarchique hlList

Au cas ou

   :(Evenement formulaire=Sur chargement)
      C_ENTIER LONG(hlList;$hSousListe;$vlDépartement;$vlEmployé;$vlDépartementID)
   ` Créer une nouvelle liste hiérarchique vide
      hlList:=Nouvelle liste
   ` Sélectionner tous les enregistrements de la table [Départements]
      TOUT SELECTIONNER([Départements])
   ` Pour chaque Département
      Boucle($vlDepartement;1;Enregistrements trouves([Départements]))
   ` Sélectionner les employés de ce département
```

```
            LIEN RETOUR([Départements]Nom)
   ` Combien sont-ils?
            $vlNbEmployés:=Enregistrements trouves([Employés])
   ` Y a-t-il au moins un employé dans ce département?
            Si($vlNbEmployés>0)
   ` Créer une sous-liste pour l'élément Département
               $hSousListe:=Nouvelle liste
   ` Pour chaque Employé
               Boucle($vlEmployé;1;Enregistrements trouves([Employés]))
   ` Ajouter l'élément Employé à la sous-liste
   ` Noter que le champ ID de l'enregistrement [Employés] est passé comme numéro de référence de l'élément
                  AJOUTER A LISTE($hSousListe;[Employés]Nom+", "+[Employés]Prénom;[Employés]ID)
   ` Aller à l'enregistrement [Employés] suivant
                  ENREGISTREMENT SUIVANT([Employés])
               Fin de boucle
            Sinon
   ` Pas d'Employé, pas de sous-liste pour l'élément Département
               $hSousListe:=0
            Fin de si

   ` Ajouter l'élément Département à la liste principale
   ` Notez que le champ ID de l'enregistrement [Départements] est passé comme numéro de référence de l'élément. Le
bit #31 de cet ID est forcé à 1.
   ` Ainsi nous pourrons faire la distinction entre les éléments Département et Employés (cf. note ci-dessous)
            AJOUTER A LISTE(hlList;[Départements]Nom;[Départements]ID?+31;$hSousListe;$hSousListe#0)
   ` Passer l'élément Département en gras pour renforcer la hiérarchie de la liste
            CHANGER PROPRIETES ELEMENT(hlList;0;Faux;Gras;0)
   ` Aller au département suivant
            ENREGISTREMENT SUIVANT([Départements])
         Fin de boucle
   ` Trier toute la liste en ordre croissant
         TRIER LISTE(hlList;>)
   ` Afficher la liste en style Windows et forcer la hauteur de ligne minimale à 14 Pts
         CHANGER PROPRIETES LISTE(hlList;A la Windows;Réf icône Windows;14)

      :(Evenement formulaire=Sur libération)
   ` La liste n'est plus utile. N'oubliez pas de l'effacer !
         SUPPRIMER LISTE(hlList;*)

      :(Evenement formulaire=Sur double clic)
   ` Il y a eu un double-clic
   ` Obtenir la position de l'élément sélectionné
         $vlÉlémentPos:=Elements selectionnes(hlList)
   ` A toutes fins utiles, vérifier la position
         Si($vlÉlémentPos # 0)
   ` Obtenir l'information de l'élément de la liste
            INFORMATION
ELEMENT(hlList;$vlÉlémentPos;$vlÉlémentRef;$vsÉlémentText;$vlÉlémentSousListe;$vbÉlémentDéployé)
   ` Cet élément est-il l'élément d'un Département?
            Si($vlÉlémentRef ?? 31)
   ` Si oui, c'est un double-clic sur un élément Département
               ALERTE("Vous avez double-cliqué sur l'élément Département
"+Caractere(34)+$vsÉlémentText+Caractere(34)+".")
            Sinon
   ` Sinon, c'est un double-clic sur un élément Employé. Avec l'ID de l'élément parent, trouver l'enregistrement
[Départements]
               $vlDépartementID:=Element parent(hlList;$vlÉlémentRef)?-31
               CHERCHER([Départements];[Départements]ID=$vlDépartementID)
   ` Indiquer où l'Employé travaille et de qui il dépend
               ALERTE("Vous avez double-cliqué sur l'élément Employé "+Caractere(34)+$vsÉlémentText+Caractere(34)+"
qui travaille dans le Département "+Caractere(34)+[Départements]Nom+Caractere(34)+" dont le responsable est
"+Caractere(34)+[Départements]Responsable+Caractere(34)+".")
            Fin de si
         Fin de si

 Fin de cas

   ` Note : 4D peut stocker jusqu'à 1 milliard d'enregistrements par table. Dans notre exemple, nous utilisons le
bit #31 de l'octet supérieur inutilisé
   ` pour différencier les éléments des Employés des Départements.
```

Dans cet exemple, il y a une seule raison d'établir une distinction entre les éléments Départements et les éléments Employés :

1. Nous stockons des ID d'enregistrements dans les numéros de référence des éléments. En conséquence, nous avons toutes les chances de rencontrer des éléments Départements dont les numéros de référence sont les mêmes que ceux des éléments Employés.

2. Nous utilisons la commande **Element parent** pour récupérer le parent de l'élément sélectionné. Si nous cliquons sur un élément Employés dont le numéro d'ID associé est 10, et s'il existe aussi un élément Départements qui a le numéro 10, l'élément Départements sera trouvé en premier par **Element parent** quand cette fonction passera la liste en revue pour repérer l'élément avec le numéro de référence que nous passons. La commande retournera le parent de l'élément Départements et non celui de l'élément Employés.

C'est pourquoi nous avons choisi des numéros de référence d'éléments uniques, non pas pour des questions de principe, mais parce que nous devions différencier les éléments de Départements et d'Employés.

Dans le formulaire en exécution, la liste apparaîtra ainsi :

Note : Cet exemple est utile dans le cadre de la gestion de l'interface utilisateur, si vous manipulez un nombre limité d'enregistrements. Souvenez-vous que les listes sont conservées en mémoire ; donc, ne construisez pas d'interfaces utilisateur exploitant des listes hiérarchiques comportant des millions d'éléments.

CHANGER ELEMENT

CHANGER ELEMENT ({* ;} liste ; refElément | * ; libelléElément ; nouvelRéf {; sous_Liste ; déployée})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Opérateur, Entier long	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
libelléElément	Chaîne	⇒	Nouveau libellé d'élément
nouvelRéf	Entier long	⇒	Nouveau numéro de référence d'élément
sous_Liste	RefListe	⇒	Nouvelle sous-liste rattachée à l'élément ou 0 = pas de sous-liste (détacher sous-liste courante) ou -1 = pas de changement
déployée	Booléen	⇒	Indique si la sous-liste doit être déployée/contractée

Description

La commande **CHANGER ELEMENT** modifie l'élément désigné par le paramètre *réfElément* de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans *réfElément*. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien.
Vous pouvez également passer *0* dans *réfElément* afin de désigner le dernier élément ajouté à la liste (à l'aide de **AJOUTER A LISTE**).
Vous pouvez enfin passer * dans *réfElément* : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Si vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la section **Gestion des listes hiérarchiques**.

Vous pouvez passer le nouveau libellé de l'élément dans le paramètre *libelléElément*. Si vous souhaitez changer le numéro de référence de l'élément, passez la nouvelle valeur dans le paramètre *nouvelRéf*, sinon passez la même valeur que dans *réfElément*.

Si vous voulez associer une sous-liste à l'élément, passez le numéro de référence de la sous-liste dans le paramètre *sous_Liste*. Dans ce cas, vous devez également spécifier si la nouvelle sous-liste devra apparaître déployée ou contractée en passant respectivement **Vrai** ou **Faux** dans le paramètre *déployée*.

Si vous voulez dissocier de l'élément une sous-liste qui lui est actuellement rattachée, passez *0* (zéro) dans *sous_Liste*. Dans ce cas, il est conseillé d'avoir préalablement obtenu le numéro de référence de cette liste à l'aide de la commande **AJOUTER A LISTE**, afin de pouvoir effacer la sous-liste avec la commande **SUPPRIMER LISTE** si vous n'en avez plus besoin.

Si vous ne souhaitez pas modifier les propriétés de sous-liste de l'élément, passez *-1* dans le paramètre *sous_Liste*.

Exemple 1

Nous supposons que *hList* est une liste dont les éléments ont des numéros de référence uniques. La méthode objet suivante d'un bouton ajoute une sous-liste à l'élément actuellement sélectionné dans la liste *hList* :

```
$vlItemPos:=Elements selectionnes(hList)
Si($vlItemPos>0)
   INFORMATION ELEMENT(hList;$vlItemPos;$vlItemRef;$vsItemText;$hSouslist;$vbExpanded)
   $vbNouvSousList:=Non(Liste existante($hSouslist))
   Si($vbNouvSousList)
      $hSouslist:=Nouvelle liste
   Fin de si
   vlUniqueRef:=vlUniqueRef+1
   AJOUTER A LISTE($hSousList;"Nouvel élément";vlUniqueRef)
   Si($vbNouvSousList)
      CHANGER ELEMENT(hList;$vlItemRef;$vsItemText;$vlItemRef;$hSouslist;Vrai)
   Fin de si
   SELECTIONNER ELEMENTS PAR REFERENCE(hList;vlUniqueRef)
Fin de si
```

Exemple 2

Reportez-vous à l'exemple de la commande **INFORMATION ELEMENT**.

Exemple 3

Reportez-vous à l'exemple de la commande **AJOUTER A LISTE**.

CHANGER PROPRIETES ELEMENT

CHANGER PROPRIETES ELEMENT ({* ;} liste ; refElément | * ; saisissable ; style ; icône {; couleur})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Opérateur, Entier long	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
saisissable	Booléen	⇒	Vrai = Saisissable, Faux = Non-saisissable
style	Entier long	⇒	Style de police pour l'élément
icône	Entier long	⇒	131072 + numéro de référence d'image ou 0
couleur	Entier long	⇒	Valeur de couleur RVB ou -1 = rétablir couleur originale

Description

La commande **CHANGER PROPRIETES ELEMENT** modifie l'élément désigné par le paramètre *réfElément* de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans *réfElément*. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien.
Vous pouvez également passer *0* dans *réfElément* afin de demander la modification du dernier élément ajouté à la liste (à l'aide de **AJOUTER A LISTE**).
Vous pouvez enfin passer * dans *réfElément* : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Si vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la section **Gestion des listes hiérarchiques**.

Note : Pour changer le libellé d'un élément ou de ses sous-listes, utilisez la commande **CHANGER ELEMENT**.

Si vous souhaitez que l'élément soit saisissable, passez **Vrai** dans le paramètre *saisissable*, sinon passez **Faux**.
Important : Pour qu'un élément soit saisissable, il doit appartenir à une liste elle-même saisissable. Pour déclarer une liste saisissable, utilisez la commande **OBJET FIXER SAISISSABLE**. La commande **CHANGER PROPRIETES ELEMENT** vous permet de déclarer un élément individuel saisissable ou non. La modification de la propriété saisissable au niveau de la liste ne change pas la propriété saisissable individuelle de chaque élément. Un élément ne peut être saisissable que si la liste **et** l'élément le sont.

Vous pouvez définir le style de l'élément dans le paramètre *styles*. Vous passez une ou une combinaison des constantes prédéfinies suivantes (thème **Styles de caractères**) :

Constante	Type	Valeur
Gras	Entier long	1
Italique	Entier long	2
Normal	Entier long	0
Souligné	Entier long	4

Si vous souhaitez associer une icône à l'élément, passez Utiliser réf image+N dans le paramètre *icône*, où N est le numéro de référence d'une image stockée dans la bibliothèque d'images de 4D, en mode Développement. Si vous ne souhaitez pas associer d'image à l'élément, passez *0* (zéro) dans *icône*.

Notes :

- Utiliser réf image est une constante prédéfinie placée dans le thème **Listes hiérarchiques**.
- Si vous souhaitez utiliser des expressions image 4D (champs, variables...) pour définir les icônes des éléments, utilisez la commande **FIXER ICONE ELEMENT**.

Le paramètre *couleur* (facultatif) permet de modifier la couleur du texte de l'élément. La couleur doit être définie sous forme de couleur RVB, c'est-à-dire un entier long de 4 octets au format 0x00RRVVBB. Pour plus d'informations sur ce format, reportez-vous à la description de la commande **OBJET FIXER COULEURS RVB**. Passez -1 dans le paramètre *couleur* pour rétablir la couleur d'origine de l'élément.

Exemple 1

Reportez-vous à l'exemple de la commande **AJOUTER A LISTE**.

Exemple 2

L'exemple suivant passe le texte de l'élément courant de *liste* en gras et en rouge vif :

```
CHANGER PROPRIETES ELEMENT(liste;*;Vrai;Gras;0;0x00FF0000)
```

CHANGER PROPRIETES LISTE

CHANGER PROPRIETES LISTE (liste ; apparence {; icône {; hauteurLigne {; doubleClic {; multiSélection {; modifiable}}}}})

Paramètre	Type		Description
liste	RefListe	⇒	Numéro de référence de la liste
apparence	Entier long	⇒	*** paramètre obsolète, toujours passer 0 ***
icône	Entier long	⇒	*** Paramètre obsolète, toujours passer 0 ***
hauteurLigne	Entier long	⇒	Hauteur minimale de la ligne (pixels)
doubleClic	Entier long	⇒	Déploiement/contraction sur double-clic 0 = autoriser, 1= empêcher
multiSélection	Entier long	⇒	Sélections multiples 0 = interdire (défaut), 1 = autoriser
modifiable	Entier long	⇒	Enumération modifiable 0 = non, 1 = oui (défaut)

Description

La commande **CHANGER PROPRIETES LISTE** définit la hauteur de ligne et le fonctionnement de la liste hiérarchique dont la référence est passée dans le paramètre *liste*.

Note de compatibilité : Les paramètres *apparence* et *icône* sont obsolètes, ils doivent toujours prendre la valeur 0.

Note : Si vous voulez personnaliser l'icône de chaque élément d'une liste hiérarchique, utilisez la commande **CHANGER PROPRIETES ELEMENT**.

Si vous ne passez pas le paramètre *hauteurLigne*, la hauteur de ligne d'une liste hiérarchique sera déterminée par la police et la taille de police utilisées pour l'objet. Vous pouvez également passer dans le paramètre *hauteurLigne* la hauteur minimale des lignes de la liste hiérarchique. Si la valeur que vous passez est supérieure à la hauteur des lignes définie par la police et la taille de police, elle sera utilisée pour fixer la hauteur des lignes. Passez 0 pour utiliser la hauteur par défaut.

Le paramètre facultatif *doubleClic* permet d'empêcher que le double-clic sur un élément de la liste ne provoque le déploiement ou la contraction de sa sous-liste.
Par défaut, une sous-liste est déployée ou contractée en cas de double-clic sur l'élément parent. Certains types d'interfaces peuvent toutefois nécessiter une désactivation de ce mécanisme. Pour cela, passez 1 dans le paramètre *doubleClic*. A noter que seul le double-clic sera désactivé. Les sous-listes pourront toujours être déployées ou contractées par un clic sur l'icône de déploiement.
Si vous passez 0 ou omettez ce paramètre, le fonctionnement par défaut est appliqué.

Le paramètre facultatif *multiSélection* permet d'indiquer si la liste doit accepter les sélections multiples.
Par défaut, il n'est pas possible de sélectionner simultanément plusieurs éléments d'une liste hiérarchique. Si vous souhaitez que cette fonction soit disponible pour la liste, passez la valeur 1 dans le paramètre *multiSélection*. Dans ce cas, les sélections multiples peuvent être effectuées :
- manuellement, à l'aide des combinaisons de touches **Maj+clic** pour une sélection continue ou **Ctrl+clic** (Windows) / **Commande+clic** (Mac OS) pour une sélection discontinue,
- par programmation, à l'aide des commandes **SELECTIONNER ELEMENTS PAR POSITION** et **SELECTIONNER ELEMENTS PAR REFERENCE**.
Si vous passez 0 ou omettez le paramètre *multiSélection*, le fonctionnement par défaut est appliqué.

Le paramètre facultatif *modifiable* permet d'indiquer si la liste sera modifiable par l'utilisateur lorsqu'elle sera affichée sous forme d'énumération associée à un champ ou une variable en saisie. Lorsque l'énumération est modifiable, un bouton **Modifier** est inséré dans la fenêtre d'énumération et l'utilisateur peut ajouter, supprimer et trier les valeurs via un éditeur spécifique.
Si vous passez 1 ou omettez le paramètre *modifiable*, l'énumération sera modifiable par l'utilisateur ; si vous passez 0, elle ne sera pas modifiable.

Exemple

Vous souhaitez interdire le déploiement/contraction sur double-clic. Vous pouvez écrire dans la méthode du formulaire :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      hlVilles:=Charger liste("Villes") //charger l'énumération Villes dans l'objet hlVilles
      CHANGER PROPRIETES LISTE(hlVilles;0;0;0;1) //pas de déploiement sur double-clic
Fin de cas
```

Charger liste

Charger liste (nomListe) -> Résultat

Paramètre	Type		Description
nomListe	Chaîne	➡	Nom de liste créée dans l'éditeur d'énumérations
Résultat	RefListe	⮌	Numéro de référence de la liste nouvellement créée

Description

La commande **Charger liste** crée une liste hiérarchique dont le contenu est copié depuis la liste *nomListe* créée en mode Développement, dans l'éditeur d'énumérations. La fonction retourne le numéro de référence de la liste nouvellement créée.

Pour connaître les énumérations définies dans la base, utilisez la commande **LISTE DES LISTES**. Pour savoir si la liste a correctement été chargée, utilisez la fonction **Liste existante** avec le numéro de référence retourné par **Charger liste**.

Notez que la nouvelle liste est une copie de la liste définie en mode Développement. Par conséquent, toute modification apportée à cette nouvelle liste n'affectera pas la liste définie en mode Développement. De même, toute modification ultérieure de l'énumération n'affecte pas la liste que vous venez de créer.

Si vous modifiez la liste nouvellement créée et voulez enregistrer ces modifications, utilisez la commande **STOCKER LISTE**.

Si vous n'avez plus besoin de la liste, n'oubliez pas d'appeler **SUPPRIMER LISTE** pour la supprimer. Sinon, elle reste en mémoire jusqu'à la fin de la session de travail ou jusqu'à ce que le process dans lequel la liste a été créée soit détruit.

Astuce : Si vous associez une liste à un objet de formulaire (liste hiérarchique, onglet ou menu hiérarchique) à l'aide du menu **Enumération** dans la Liste des propriétés, il est inutile d'appeler **Charger liste** ou **SUPPRIMER LISTE** dans la méthode de l'objet. 4D charge et efface la liste automatiquement pour vous.

Exemple

Imaginons que vous créez une base pour le marché international. Vous voulez pouvoir changer la langue utilisée. Dans un formulaire, vous présentez une liste hiérarchique *listeHL* qui propose les langues disponibles. En mode Développement, vous avez préparé des listes différentes, par exemple “Options US” pour la version anglaise, “Options FR” pour la version française, “Options ES” pour la version espagnole, etc. De plus, vous maintenez la variable interprocess *<>gaLangueCourante* dans laquelle vous stockez un code de langue sur 2 caractères, par exemple “US” pour la version anglaise, “FR” pour la version française, “ES” pour la version espagnole, etc. Pour vous assurer que la liste correcte sera chargée en utilisant la langue choisie, vous pouvez écrire :

```
   ` Méthode objet de la liste hiérarchique listeHL
Au cas ou
   :(Evenement formulaire=Sur chargement)
      C_ENTIER LONG(listeHL)
      listeHL:=Charger liste("Options"+<>gaLangueCourante)
   :(Evenement formulaire=Sur libération)
      SUPPRIMER LISTE(listeHL;*)
Fin de cas
```

Chercher dans liste

Chercher dans liste ({* ;} liste ; valeur ; portée {; tabEléments {; *}}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
valeur	Chaîne	⇒	Valeur à rechercher
portée	Entier	⇒	0=Liste principale, 1=Sous-listes
tabEléments	Tableau entier long	⇐	- Si 2e * omis : tableau des positions des éléments trouvés - Si 2e * passé : tableau des numéros de référence des éléments trouvés
*	Opérateur	⇒	- Si omis : utiliser la position des éléments - Si passé : utiliser le numéro de référence des éléments
Résultat	Entier long	⇐	- Si 2e * omis : position de l'élément trouvé - Si 2e * passé : numéro de référence de l'élément trouvé

Description

La commande **Chercher dans liste** retourne la position ou la référence du premier élément de *liste* qui équivaut à la chaîne passée dans *valeur*. Si plusieurs éléments sont trouvés, la fonction peut également remplir le tableau *tabEléments* avec la position ou la référence de chaque élément.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les numéros de référence des éléments (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec les positions des éléments (le second * est passé), la syntaxe basée sur le nom d'objet est requise car la position des éléments peut varier d'une représentation à l'autre.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **Chercher dans liste** s'appliquera au premier objet dont le nom correspond.

Le second paramètre * vous permet d'indiquer si vous souhaitez travailler avec des positions courantes d'éléments (dans ce cas, ce paramètre doit être omis) ou des références absolues d'éléments (dans ce cas, il doit être passé).

Passez dans *valeur* la chaîne de caractères à rechercher. La recherche sera du type "est exactement", c'est-à-dire que la recherche de "bois"ne trouvera pas "boissons". Toutefois, vous pouvez utiliser le caractère @ pour définir des recherches du type "commence par", "se termine par" ou "contient".

Le paramètre *portée* vous permet de définir si la recherche doit porter uniquement sur le premier niveau de la *liste* ou si elle doit inclure toutes ses sous-listes. Passez 0 pour concentrer la recherche sur le premier niveau de la liste et 1 pour l'étendre à toutes les sous-listes.

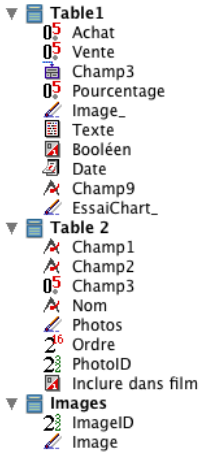
Si vous souhaitez connaître la position ou le numéro de tous les éléments correspondant à *valeur*, passez un tableau d'entiers longs dans le paramètre facultatif *tabEléments*. Si nécessaire, le tableau sera créé et redimensionné par la commande. La commande remplira le tableau avec les positions (si le second * est omis) ou les numéros de référence (si le second * est passé) des éléments trouvés.

Les positions sont exprimées relativement à l'élément supérieur de la liste principale, en tenant compte de l'état courant déployé/contracté de la liste et de ses sous-listes.

Si aucun élément ne correspond à la *valeur* recherchée, la fonction retourne 0 et le tableau *tabEléments* est retourné vide.

Exemple

Soit la liste hiérarchique suivante :




```
$vlItemPos:=Chercher dans liste(hList;"P@";1;$arrPos)
  `$vlItemPos vaut 5
  `$arrPos{1} vaut 5, $arrPos{2} vaut 17 et $arrPos{3} vaut 19
$vlItemRef:=Chercher dans liste(hList;"P@";1;$arrRefs;*)
  `$vlItemRef vaut 7
  `$arrRefs{1} vaut 7, $arrRefs{2} vaut 18 et $arrRefs{3} vaut 23
$vlItemPos:=Chercher dans liste(hList;"Date";1;$arrPos)
  `$vlItemPos vaut 9
  `$arrPos{1} vaut 9
$vlItemRef:=Chercher dans liste(hList;"Date";1;$arrRefs;*)
  `$vlItemRef vaut 11
  `$arrRefs{1} vaut 11
$vlItemPos:=Chercher dans liste(hList;"Date";0;*)
  `$vlItemPos vaut 0`
```

Copier liste

Copier liste (liste) -> Résultat

Paramètre	Type		Description
liste	RefListe	⇒	Numéro de référence de la liste à copier
Résultat	RefListe	⇐	Numéro de référence de la nouvelle liste

Description

La commande **Copier liste** duplique la liste dont vous passez le numéro de référence dans le paramètre *liste* et retourne le numéro de référence de la nouvelle liste.

Le contenu de la liste copiée est entièrement dupliqué. Une fois que vous en avez terminé avec la copie de la liste, appelez la commande **SUPPRIMER LISTE** pour l'effacer.

Element parent

Element parent ({* ;} liste ; refElément | *) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Opérateur, Entier long	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
Résultat	Entier long	⇐	Numéro de référence de l'élément parent ou 0 s'il n'y en a pas

Description

La commande **Element parent** retourne le numéro de référence de l'élément parent.

Passez dans *liste* le numéro de référence ou le nom d'objet de la liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **Element parent** s'appliquera au premier objet dont le nom correspond.

Passez dans *réfElément* le numéro de référence d'un élément de la liste ou 0, ou encore *. Si vous passez 0, la commande s'applique au dernier élément ajouté à la liste. Si vous passez *, la commande s'applique à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés, l'élément courant est celui qui a été sélectionné en dernier.

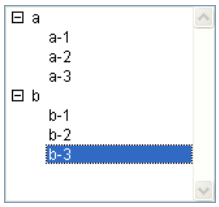
En retour, si un élément correspondant existe bien dans la liste et si cet élément se trouve bien dans une sous-liste (et a donc un élément parent), vous récupérez le numéro de référence de l'élément parent.

S'il n'existe pas d'élément numéro *réfElément*, ou si vous avez passé * et qu'aucun élément n'est sélectionné, ou si cet élément n'a pas d'élément parent, **Element parent** retourne *0* (zéro).

Si vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la description de la commande **AJOUTER A LISTE**.

Exemple

Voici une liste *hList* affichée en mode Application :



Voici les numéros de référence des éléments de cette liste :

Elément	Numéro
a	100
a-1	101
a-2	102
b	200
b-1	201
b-2	202
b-3	203

- Avec le code ci-dessous, si l'élément "b-3" est sélectionné, la variable *$vlParentElémRef* prend la valeur 200, c'est-à-dire le numéro de référence de l'élément "b" :

```
$vlElémPos:=Elements selectionnes(hList)
INFORMATION ELEMENT(hList;$vlElémPos;$vlElémRef;$vsItemText)
$vlParentElémRef:=Element parent(hList;$vlElémRef)  ` $vlParentElémRef vaut 200
```

- Si l'élément "a-1" était sélectionné, la variable *$vlParentElémRef* prendrait la valeur 100, c'est-à-dire le numéro de référence de l'élément "a".
- Si l'élément "a" ou "b" était sélectionné, la variable *$vlParentElémRef* prendrait la valeur 0 car ces éléments n'ont pas d'élément parent.

Elements selectionnes

Elements selectionnes ({* ;} liste {; tabEléments {; *}}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
tabEléments	Tableau entier long	⇐	Si 2e * omis : Tableau des positions des éléments sélectionnés dans la ou les liste(s) Si 2e * passé : Tableau des références des éléments sélectionnés dans la ou les liste(s)
*	Opérateur	⇒	Si omis : Position(s) d'élément(s) Si passé : Référence(s) d'élément(s)
Résultat	Entier long	⇐	Si 2e * omis : Position de l'élément sélectionné parmi la ou les liste(s) déployée(s)/contractée(s) Si 2e * passé : Référence de l'élément sélectionné

Description

La fonction **Elements selectionnes** retourne la **position** ou la **référence** de l'élément sélectionné dans la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les références d'éléments (le second * est passé), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec les positions des éléments (le second * est omis), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration d'éléments déployés/contractés.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **Elements selectionnes** s'appliquera au premier objet dont le nom correspond.

En cas de sélection multiple, la fonction peut également retourner dans le tableau *tabEléments* la position ou la référence de chaque élément sélectionné. Cette fonction doit être appliquée à une liste affichée dans un formulaire afin de détecter le ou les élément(s) sélectionné(s) par l'utilisateur.

Le second paramètre * permet d'indiquer si vous souhaitez travailler avec des positions courantes d'éléments (dans ce cas, ce paramètre doit être omis) ou des références absolues d'éléments (dans ce cas, il doit être passé).

Vous pouvez passer dans le paramètre *tabEléments* un tableau d'entiers longs. Si nécessaire, le tableau sera créé et redimensionné par la commande. A l'issue de l'exécution de la commande, *tabEléments* contiendra :

- la position de chaque élément sélectionné relativement à l'état déployé/contracté de la ou des liste(s) si le paramètre * est omis.
- la référence absolue de chaque élément sélectionné si le paramètre * est passé.
  Le tableau est retourné vide si aucun élément n'est sélectionné.

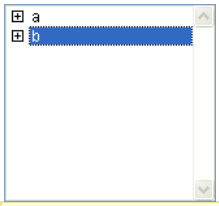
Note : En cas de sélection multiple, la commande retourne la position ou la référence de l'élément courant de *liste*. L'élément courant est le dernier élément sur lequel l'utilisateur a cliqué (sélection manuelle) ou l'élément désigné par la commande **SELECTIONNER ELEMENTS PAR POSITION** ou **Elements selectionnes** (sélection par programmation).

Si la liste comporte des sous-listes, appliquez la fonction à la liste principale (celle qui est associée au formulaire), et non à une de ses sous-listes. Les positions sont exprimées relativement à l'élément supérieur de la liste principale, en tenant compte de l'état courant déployé/contracté de la liste et de ses sous-listes.

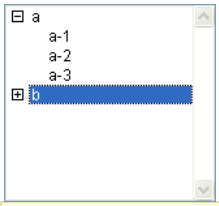
Dans tous les cas, si aucun élément n'est sélectionné, la fonction retourne 0.

Exemple

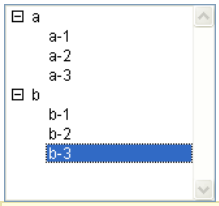
Voici la liste *hList* telle qu'elle apparaît en mode Application :



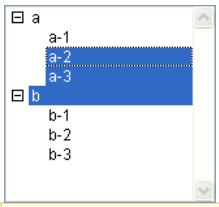

```
$vlItemPos:=Elements selectionnes(hList) ` à ce stade, $vlItemPos vaut 2
```



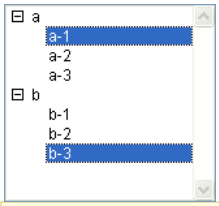

```
$vlItemPos:=Elements selectionnes(hList) ` à ce stade, $vlItemPos vaut 5
$vlItemRef:=Elements selectionnes(hList;*) ` $vlItemRef vaut 200 (par exemple)
```




```
$vlItemPos:=Elements selectionnes(hList) ` à ce stade, $vlItemPos vaut 8
$vlItemRef:=Elements selectionnes(hList;*) ` $vlItemRef vaut 203 (par exemple)
```




```
$vlItemPos:=Elements selectionnes(hList;$tabPos) ` à ce stade, $vlItemPos vaut 3
 ` $tabPos{1} vaut 3, $tabPos{2} vaut 4 et $tabPos{3} vaut 5
```




```
$vlItemRef:=Elements selectionnes(hList;$tabRefs;*) ` $vlItemRef vaut 203 (par exemple)
 ` $tabRefs{1} vaut 101, $tabRefs{2} vaut 203 (par exemple)
```

FIXER ICONE ELEMENT

FIXER ICONE ELEMENT ({* ;} liste ; réfElément | * ; icône)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long, Opérateur	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
icône	Image	⇒	Icône à associer à l'élément

Description

La commande **FIXER ICONE ELEMENT** permet de modifier l'icône associée à l'élément désigné par le paramètre *réfElément* de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Note : Il est possible de modifier l'icône associée à un élément à l'aide de la commande **CHANGER PROPRIETES ELEMENT**. Toutefois, **CHANGER PROPRIETES ELEMENT** accepte uniquement des références d'images statiques (références de ressources ou images de la bibliothèque).

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans *réfElément*. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien. Vous pouvez également passer 0 dans *réfElément* afin de demander la modification du dernier élément ajouté à la liste (à l'aide de **AJOUTER A LISTE**).
Vous pouvez enfin passer * dans *réfElément* : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Passez dans le paramètre *icône* une expression image 4D valide (champ, variable, pointeur, etc.). L'image sera placée à gauche de l'élément.

Exemple

Affectation d'une même image à deux éléments différents. Ce code est optimisé car l'image est chargée une seule fois en mémoire :

```
C_IMAGE($image)
LIRE FICHIER IMAGE("monImage.png";$image)
FIXER ICONE ELEMENT(maliste;ref1;$image)
FIXER ICONE ELEMENT(maliste;ref2;$image)
```

FIXER PARAMETRE ELEMENT

FIXER PARAMETRE ELEMENT ({* ;} liste ; refElément | * ; sélecteur ; valeur)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Opérateur, Entier long	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
sélecteur	Chaîne	⇒	Constante de paramètre
valeur	Chaîne, Booléen, Entier long, Réel	⇒	Valeur de paramètre

Description

La commande **FIXER PARAMETRE ELEMENT** permet de modifier le paramètre *sélecteur* pour l'élément *réfElément* de la liste hiérarchique dont vous avez passé la référence ou le nom d'objet dans le paramètre *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans *réfElément*. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien.Vous pouvez également passer 0 dans *réfElément* afin de demander la modification du dernier élément ajouté à la liste (à l'aide de **Listes hiérarchiques**). Vous pouvez enfin passer * dans *réfElément* : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Vous pouvez passer dans *sélecteur* la constante Texte supplémentaire (placée dans le thème "**Listes hiérarchiques**") ou toute valeur personnalisée :

- Texte supplémentaire : cette constante permet d'ajouter un texte à droite de l'élément *réfElément*. Ce libellé supplémentaire reste toujours affiché dans la partie droite de la liste, même si l'utilisateur déplace le curseur de défilement horizontal. Lorsque vous utilisez cette constante, passez dans *valeur* le texte à afficher.
- Sélecteur personnalisé : vous pouvez passer dans *sélecteur* tout texte personnalisé et lui associer une valeur de type texte, numérique ou booléen. Cette valeur sera stockée avec l'élément et pourra être récupérée via la commande **LIRE PARAMETRE ELEMENT**. Ce principe permet de mettre en place tout type d'interface associée aux listes hiérarchiques. Par exemple, dans une liste stockant des noms de personnes, vous pouvez stocker l'âge de chaque personne et ne l'afficher que lorsque l'élément correspondant est sélectionné.

FIXER POLICE ELEMENT

FIXER POLICE ELEMENT ({* ;} liste ; refElément | * ; police)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Entier long, Opérateur	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
police	Chaîne, Entier long	⇒	Nom ou numéro de police

Description

La commande **FIXER POLICE ELEMENT** modifie la police de caractères de l'élément désigné par le paramètre *réfElément* de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans *réfElément*. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien.Vous pouvez également passer 0 dans *réfElément* afin de demander la modification du dernier élément ajouté à la liste (à l'aide de **AJOUTER A LISTE**). Vous pouvez enfin passer * dans *réfElément* : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Passez dans le paramètre *police* le nom ou le numéro de la police à utiliser. Pour réappliquer la police par défaut de la liste hiérarchique, passez une chaîne vide dans *police*.

Exemple

Appliquer la police Times à l'élément courant de la liste :

```
FIXER POLICE ELEMENT(*;"Maliste";*;"Times")
```

INFORMATION ELEMENT

INFORMATION ELEMENT ({* ;} liste ; positionElém | * ; réfElément ; libelléElément {; sous_Liste ; déployée})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
positionElém | *	Opérateur, Entier long	⇒	Position de l'élément dans la ou les liste(s) déployée(s)/contractée(s) ou * pour l'élément courant de la liste
réfElément	Entier long	⇐	Numéro de référence de l'élément
libelléElément	Chaîne	⇐	Libellé de l'élément
sous_Liste	RefListe	⇐	Numéro de référence de sous-liste (s'il y en a)
déployée	Booléen	⇐	Si une sous-liste est rattachée à l'élément : Vrai = la sous-liste est déployée Faux = la sous-liste est contractée

Description

La commande **INFORMATION ELEMENT** retourne des informations sur l'élément désigné par le paramètre *positionElém* de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste, vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste, la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration déployée/contractée et de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **INFORMATION ELEMENT** s'appliquera au premier objet dont le nom correspond.

La position doit être exprimée relativement à l'état déployé/contracté de la liste et de ses sous-listes. Vous devez passer une valeur de position comprise entre 1 et la valeur retournée par **Nombre elements**. Si vous passez une valeur située hors de cet intervalle, **INFORMATION ELEMENT** retourne des valeurs vides (0, "", etc.).
Si vous passez * dans *positionElém*, la commande s'applique à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande retourne des valeurs vides.

Après l'appel, vous récupérez :

- Le numéro de référence de l'élément dans *réfElément*.
- Le libellé de l'élément dans *libelléElém*.

Si vous passez les paramètres optionnels *sous_Liste* et *déployée* :

- *sous_Liste* contient le numéro de référence de la sous-liste rattachée à l'élément. Si l'élément n'a pas de sous-liste associée, *sous_Liste* retourne zéro.
- Si l'élément comporte une sous-liste, *déployée* retourne **Vrai** si la sous-liste est déployée, et **Faux** sinon.

Exemple 1

En partant de l'hypothèse que *hList* est une liste dont les éléments ont des numéros de référence uniques, le code suivant inverse automatiquement l'état déployé/contracté de la sous-liste, si elle existe, rattachée à l'élément sélectionné :

```
C_BOOLEEN($vbDéployé)
C_ENTIER LONG($hSousListe;$vlElemRef)
C_ALPHA(31;$vsElemText)
 `La déclaration de ces variables est nécessaire si vous souhaitez compiler la méthode

$vlElemPos:=Elements selectionnes(hList)
Si($vlElemPos>0)
   INFORMATION ELEMENT(hList;$vlElemPos;$vlElemRef;$vsElemText;$hSousListe;$vbDéployé)
   Si(Liste existante($hSousListe))
      CHANGER ELEMENT(hList;$vlElemRef;$vsElemText;$hSousListe;Non($vbDéployé))
   Fin de si
Fin de si
```

Exemple 2

Reportez-vous à l'exemple de la commande **AJOUTER A LISTE**.

INSERER DANS LISTE

INSERER DANS LISTE ({* ;} liste ; avantElément | * ; libelléElément ; réfElément {; sous_Liste ; déployée})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
avantElément | *	Entier long, Opérateur	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément de la liste actuellement sélectionné
libelléElément	Chaîne	⇒	Libellé du nouvel élément
réfElément	Entier long	⇒	Numéro de référence unique du nouvel élément
sous_Liste	RefListe	⇒	Sous-liste optionnelle rattachée au nouvel élément
déployée	Booléen	⇒	Indique si la sous-liste doit être déployée ou non

Description

La commande **INSERER DANS LISTE** insère l'élément désigné par le paramètre *réfElément* dans la liste dont le numéro de référence ou le nom d'objet est passé dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Le paramètre *avantElément* permet de désigner l'élément avant lequel vous souhaitez insérer le nouvel élément :

- Vous pouvez passer la valeur 0 afin de désigner le dernier élément ajouté à la liste. Le nouvel élément devient l'élément sélectionné.
- Vous pouvez passer * afin que le nouvel élément soit inséré avant l'élément actuellement sélectionné dans la liste. Le nouvel élément devient l'élément sélectionné.
- Si vous souhaitez insérer le nouvel élément avant un élément spécifique, passez le numéro de référence de cet élément comme deuxième paramètre. Dans ce cas, le nouvel élément inséré n'est pas automatiquement sélectionné. Si le numéro que vous passez ne correspond à aucun élément de la *liste*, la commande ne fait rien.

Vous passez le texte et le numéro de référence du nouvel élément dans les paramètres *libelléElément* et *réfElément*.

Si vous souhaitez que l'élément comporte des sous-éléments, passez un numéro de référence de liste valide dans le paramètre *sous_Liste*. Dans ce cas, vous devez également passer le paramètre *déployée*. Passez **Vrai** ou **Faux** dans ce paramètre pour que cette sous-liste s'affiche respectivement déployée ou contractée.

Exemple

L'exemple suivant insère un élément (associé à aucune sous-liste) juste devant l'élément actuellement sélectionné dans la liste *hList*:

```
vlUniqueRef:=vlUniqueRef+1
INSERER DANS LISTE(hList;*;"Nouvel élément";vlUniqueRef)
```

LIRE ICONE ELEMENT

LIRE ICONE ELEMENT ({* ;} liste ; refElément | * ; icône)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Opérateur, Entier long	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
icône	Variable image	⇐	Icône associée à l'élément

Description

La commande **LIRE ICONE ELEMENT** retourne dans *icône* l'icône associée à l'élément dont vous avez passé le numéro de référence dans *réfElément* de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **LIRE ICONE ELEMENT** s'appliquera au premier objet dont le nom correspond.

Vous pouvez passer un numéro de référence dans *réfElément*. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien. Vous pouvez également passer 0 dans *réfElément* afin de désigner le dernier élément ajouté à la liste (à l'aide de **AJOUTER A LISTE**).
Vous pouvez enfin passer * dans *réfElément* : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Passez dans *icône* une variable image. A l'issue de l'exécution de la commande, elle contiendra l'icône associée à l'élément, quelle que soit la source de l'icône (image statique, ressource ou expression image).

Si aucune icône n'est associée à l'élément, la variable icône est retournée vide.

Note : Lorsque l'icône associée à un élément a été définie via une référence statique (références de ressources ou images de la bibliothèque), il est possible de connaître son numéro à l'aide de la commande **LIRE PROPRIETES ELEMENT**.

LIRE PARAMETRE ELEMENT

LIRE PARAMETRE ELEMENT ({* ;} liste ; refElément | * ; sélecteur ; valeur)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Opérateur, Entier long	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
sélecteur	Chaîne	⇒	Constante de paramètre
valeur	Chaîne, Booléen, Réel	⇐	Valeur courante du paramètre

Description

La commande **LIRE PARAMETRE ELEMENT** permet de connaître la *valeur* courante du paramètre *sélecteur* pour l'élément *réfElément* de la liste hiérarchique dont vous avez passé la référence ou le nom d'objet dans le paramètre *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **LIRE PARAMETRE ELEMENT** s'appliquera au premier objet dont le nom correspond.

Vous pouvez passer un numéro de référence dans *réfElément*. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien. Vous pouvez également passer 0 dans *réfElément* afin de désigner le dernier élément ajouté à la liste (à l'aide de **AJOUTER A LISTE**).
Vous pouvez enfin passer * dans *réfElément* : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Vous pouvez passer dans *sélecteur* la constante Texte supplémentaire (placée dans le thème "**Listes hiérarchiques**") ou toute valeur personnalisée. Pour plus d'informations sur les paramètres *sélecteur* et *valeur*, reportez-vous à la description de la commande **FIXER PARAMETRE ELEMENT**.

LIRE PARAMETRE ELEMENT TABLEAUX

LIRE PARAMETRE ELEMENT TABLEAUX ({* ;} liste ; refElément | * ; tabSélecteurs {; tabValeurs})

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RefListe, Chaîne	➡	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Entier long, Opérateur	➡	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
tabSélecteurs	Tableau texte	⬅	Tableau des noms de paramètres
tabValeurs	Tableau texte	⬅	Tableau des valeurs de paramètres

Description

La commande **LIRE PARAMETRE ELEMENT TABLEAUX** permet de récupérer en un seul appel l'ensemble des paramètres (ainsi que, optionnellement, leurs valeurs) associés à l'élément *refElément* de la liste hiérarchique dont vous avez passé la référence ou le nom d'objet dans le paramètre *liste*.

Les paramètres associés aux éléments permettent de stocker des informations supplémentaires sur chaque élément. Ils sont définis à l'aide de la commande **FIXER PARAMETRE ELEMENT**.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RefListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

LIRE PARAMETRE ELEMENT TABLEAUX retourne les paramètres définis pour l'élément *réfElément* dans le tableau texte *tabSélecteurs*. Si le tableau texte *tabValeurs* est passé, la commande retourne les valeurs associées à chaque paramètre dans ce tableau.

Le tableau *tabValeur* doit être de type texte. Si vous avez associé des valeurs non-textuelles (type numérique ou booléen), elles sont converties en chaînes (vrai="1", faux="0").

Exemple

Soit la liste hiérarchique suivante :

```
<>HL:=Nouvelle liste
$ID:=30
AJOUTER A LISTE(<>HL;"Martin";$ID)
    //5 paramètres
FIXER PARAMETRE ELEMENT(<>HL;$ID;"Firstname";"Phil")
FIXER PARAMETRE ELEMENT(<>HL;$ID;"Birthday";"15/02/1978")
FIXER PARAMETRE ELEMENT(<>HL;$ID;"Male";Vrai)   //booléen
FIXER PARAMETRE ELEMENT(<>HL;$ID;"Age";32)   //numérique
FIXER PARAMETRE ELEMENT(<>HL;$ID;"City";"Nantes")
```

Pour plus de simplicité, la liste a été associée à un objet liste de même nom ("<>HL").
Lorsque l'élément "Martin" est sélectionné dans la liste, on peut lire ses paramètres en exécutant le code suivant :

```
TABLEAU TEXTE(tNomsParams;0)
LIRE PARAMETRE ELEMENT TABLEAUX(*;"<>HL";*;tNomsParams)
    // tNomsParams{1} contient "Firstname"
    // tNomsParams{2} contient "Birthday"
    // tNomsParams{3} contient "Male"
    // tNomsParams{4} contient "Age"
    // tNomsParams{5} contient "City"
```

Si on souhaite récupérer également les valeurs des paramètres, on peut écrire :

```
TABLEAU TEXTE(tNomsParams;0)
TABLEAU TEXTE(tValeursParams;0)
LIRE PARAMETRE ELEMENT TABLEAUX(*;"<>HL";*;tNomsParams;tValeursParams)
    // tValeursParams{1} contient "Phil"
    // tValeursParams{2} contient "15/02/1978"
    // tValeursParams{3} contient "1"
    // tValeursParams{4} contient "32"
    // tValeursParams{5} contient "Nantes"
```

Lire police element

Lire police element ({* ;} liste ; refElément | *) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Entier long, Opérateur	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
Résultat	Chaîne	⇐	Nom de police

Description

La commande **Lire police element** retourne le nom de la police de caractères courante de l'élément désigné par le paramètre *réfElément* de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **Lire police element** s'appliquera au premier objet dont le nom correspond.

Vous pouvez passer un numéro de référence dans *réfElément*. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien. Vous pouvez également passer 0 dans *réfElément* afin de désigner le dernier élément ajouté à la liste (à l'aide de **AJOUTER A LISTE**).

Vous pouvez enfin passer * dans *réfElément* : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

LIRE PROPRIETES ELEMENT

LIRE PROPRIETES ELEMENT ({* ;} liste ; refElément | * ; saisissable {; style {; icône {; couleur}}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Opérateur, Entier long	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
saisissable	Booléen	⇐	Vrai = Saisissable, Faux = Non-saisissable
style	Entier long	⇐	Style de police de l'élément
icône	Entier long	⇐	131072 + numéro de référence d'image
couleur	Entier long	⇐	Valeur de couleur RVB

Description

La commande **LIRE PROPRIETES ELEMENT** retourne les propriétés de l'élément désigné par le paramètre *réfElément* de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **LIRE PROPRIETES ELEMENT** s'appliquera au premier objet dont le nom correspond.

Vous pouvez passer dans *réfElément* un numéro de référence, la valeur 0 afin de désigner le dernier élément ajouté à la liste ou encore * afin de désigner l'élément courant de la liste. Si plusieurs éléments sont sélectionnés, l'élément courant est celui qui a été sélectionné en dernier.
Si vous passez * et qu'aucun élément n'est sélectionné ou si le numéro de référence d'élément ne correspond à aucun élément de la liste, la commande laisse les paramètres inchangés.

Si vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la description de la commande **AJOUTER A LISTE**.

Après l'appel :

- *saisissable* retourne Vrai si l'élément est saisissable.
- *style* retourne le style de caractères de l'élément.
- *icône* retourne l'icône ou l'image associée à l'élément, et *0* s'il n'y en a pas.
- *couleur* retourne la couleur du texte de l'élément désigné.

Note : Vous pouvez récupérer dans une variable image l'icône associée à un élément à l'aide de la commande **LIRE ICONE ELEMENT**.

Pour plus d'informations sur ces propriétés, reportez-vous à la description de la commande **CHANGER PROPRIETES ELEMENT**.

LIRE PROPRIETES LISTE

LIRE PROPRIETES LISTE (liste ; apparence {; icône {; hauteurLigne {; doubleClic {; multiSélection {; modifiable}}}}})

Paramètre	Type		Description
liste	RefListe	⇒	Numéro de référence de la liste
apparence	Entier long	⇐	Style graphique de la liste 1 = Liste hiérarchique à la Macintosh 2 = Liste hiérarchique à la Windows
icône	Entier long	⇐	*** Paramètre obsolète, retourne 0 ***
hauteurLigne	Entier long	⇐	Hauteur minimale de la ligne (pixels)
doubleClic	Entier long	⇐	Déploiement/contraction sur double-clic 0 = autorisé, 1= empêché
multiSélection	Entier long	⇐	Sélections multiples : 0 = interdites, 1 = autorisées
modifiable	Entier long	⇐	Enumération modifiable : 0 = non, 1 = oui

Description

La commande **LIRE PROPRIETES LISTE** retourne des informations sur la liste hiérarchique dont vous avez passé le numéro de référence dans le paramètre *liste*.

Le paramètre *apparence* retourne le style graphique de la liste.
Le paramètre *icône* est obsolète, il retourne toujours 0.
Le paramètre *hauteurLigne* retourne la hauteur de ligne minimale.
Si le paramètre *doubleClic* vaut 1, le déploiement ou la contraction des sous-listes en cas de double-clic sur l'élément parent est désactivé(e). Si *doubleClic* vaut 0, ce fonctionnement est actif (valeur par défaut).
Si le paramètre *multiSélection* vaut 0, la sélection multiple d'éléments (manuelle ou par programmation) n'est pas possible dans la liste. S'il vaut 1, la sélection multiple est permise.
Si le paramètre *modifiable* vaut 1, la liste est modifiable lorsqu'elle est affichée sous forme d'énumération dans les enregistrements. S'il vaut 0, la liste n'est pas modifiable.

Ces propriétés peuvent être définies à l'aide de la commande **CHANGER PROPRIETES LISTE** et/ou dans l'éditeur d'énumérations en mode Développement, si la liste a été créée dans cet éditeur ou sauvegardée avec la commande **STOCKER LISTE**.

Pour une description complète de ces propriétés d'apparence et de comportement, reportez-vous à la commande **CHANGER PROPRIETES LISTE**.

LISTE DES LISTES

LISTE DES LISTES (tabNums ; tabNoms)

Paramètre	Type		Description
tabNums	Tableau entier long	⇦	Numéros des énumérations
tabNoms	Tableau texte	⇦	Noms des énumérations

Description

La commande **LISTE DES LISTES** retourne dans les tableaux synchronisés *tabNums* et *tabNoms* les numéros et les noms des énumérations définies dans l'éditeur d'énumérations en mode Développement.

Les numéros des énumérations correspondent à leur ordre de création. Dans l'éditeur d'énumérations, les énumérations sont affichées par ordre alphabétique.

Liste existante

Liste existante (liste) -> Résultat

Paramètre	Type		Description
liste	RefListe	⇒	Référence de la liste à tester
Résultat	Booléen	⇐	Vrai si liste est une liste hiérarchique Faux si liste n'est pas une liste hiérarchique

Description

La fonction **Liste existante** retourne **VRAI** si la valeur passée dans le paramètre *liste* est une référence valide à une liste hiérarchique. Dans les autres cas, elle retourne **FAUX**.

Exemple 1

Reportez-vous à l'exemple de la commande **SUPPRIMER LISTE**.

Exemple 2

Reportez-vous aux exemples de la commande **PROPRIETES GLISSER DEPOSER**.

Nombre elements

Nombre elements ({* ;} liste {; *}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
*	Opérateur	⇒	Si omis (défaut) : Retourner les éléments visibles (déployés) dans la ou les liste(s) Si spécifié : Retourner tous les éléments
Résultat	Entier long	⇐	Nombre d'éléments visibles (déployés) si 2e * omis ou Nombre total d'éléments si 2e * passé

Description

La fonction **Nombre elements** retourne soit le nombre d'éléments visibles soit le nombre total d'éléments dans la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec tous les éléments (le second * est passé), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec les éléments visibles (le second * est omis), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration déployée/contractée.

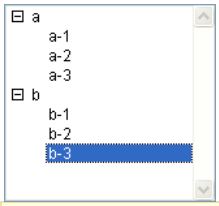
Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **Nombre elements** s'appliquera au premier objet dont le nom correspond.

Le choix du type d'information à retourner est effectué à l'aide du second paramètre *. Lorsque ce paramètre est passé, la commande retourne le nombre total d'éléments présents dans la liste, quel que soit son état courant déployé/contracté.
Lorsque ce paramètre est omis, la commande retourne le nombre d'éléments qui sont visibles, en fonction de l'état déployé/contracté actuel de la liste et de ses sous-listes.

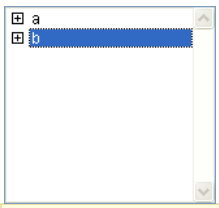
Cette fonction doit être appliquée à une liste affichée dans un formulaire.

Exemples

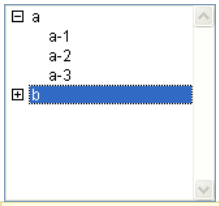
Voici la liste *hList* affichée en mode Application :




```
$vlNbItems:=Nombre elements(hList) ` à ce stade, $vlNbItems vaut 8
$vlNbTItems:=Nombre elements(hList;*) `$vlNbTItems vaut également 8
```




```
$vlNbItems:=Nombre elements(hList) ` à ce stade, $vlNbItems vaut 2
$vlNbTItems:=Nombre elements(hList;*) `$vlNbTItems vaut toujours 8
```




```
$vlNbItems:=Nombre elements(hList) `$vlNbItems vaut 5
$vlNbTItems:=Nombre elements(hList;*) `$vlNbTItems vaut toujours 8
```

Nouvelle liste

Nouvelle liste -> Résultat

Paramètre	Type		Description
Résultat	RefListe		Numéro de référence de liste

Description

La commande **Nouvelle liste** crée une nouvelle liste hiérarchique vide en mémoire et retourne son numéro de référence unique.

ATTENTION : Les listes hiérarchiques résident en mémoire. Une fois que vous en avez terminé avec une liste hiérarchique, il est important que vous l'effaciez à l'aide de la commande **SUPPRIMER LISTE**. Ainsi, vous libérez la mémoire occupée par la liste hiérarchique dont vous n'avez plus besoin.

D'autres commandes vous permettent de créer des listes hiérarchiques :

- **Copier liste** crée une nouvelle liste en dupliquant une liste existante.
- **Charger liste** crée une nouvelle liste en chargeant une énumération créée (manuellement ou par programmation) dans l'éditeur d'énumérations du mode Développement.
- **BLOB vers liste** crée une nouvelle liste à partir du contenu d'un BLOB dans lequel une liste avait été préalablement stockée.

Une fois que vous avez créé une liste hiérarchique à l'aide de la commande **Nouvelle liste**, vous pouvez :

- Ajouter des éléments à la liste à l'aide des commandes **AJOUTER A LISTE** et **INSERER DANS LISTE**.
- Supprimer des éléments de cette liste à l'aide de la commande **SUPPRIMER DANS LISTE**.

Exemple

Reportez-vous à l'exemple de la commande **AJOUTER A LISTE**.

Position element liste

Position element liste ({* ;} liste ; réfElément) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément	Entier long	⇒	Numéro de référence d'élément
Résultat	Entier long	⇐	Position de l'élément parmi la ou les liste(s) déployée(s)/contractée(s)

Description

La commande **Position element liste** retourne la position de l'élément dont vous avez passé le numéro de référence dans *réfElément* parmi la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste, vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste, la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration déployée/contractée.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **Position element liste** s'appliquera au premier objet dont le nom correspond.

Note : A la différence des autres commandes de ce thème, cette commande ne permet pas de passer la valeur 0 dans *réfElément* pour désigner le dernier élément ajouté.

La position est exprimée relativement à l'élément supérieur de la liste, en tenant compte de l'état déployé/contracté de la liste et de ses sous-listes.

Le résultat est donc compris entre 1 et la valeur retournée par **Nombre elements**.

Si l'élément n'est pas visible car il est inclus dans une liste contractée, **Position element liste** déploie la liste correspondante de manière à ce que l'élément devienne visible.

Si l'élément n'existe pas, **Position element liste** retourne *0*.

SELECTIONNER ELEMENTS PAR POSITION

SELECTIONNER ELEMENTS PAR POSITION ({* ;} liste ; positionElém {; tabPositions})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
positionElém	Entier long	⇒	Position de l'élément dans la ou les liste(s) déployée(s)/contractée(s)
tabPositions	Tableau entier long	⇒	Tableau de positions dans la ou les liste(s) déployée(s)/contractée(s)

Description

La commande **SELECTIONNER ELEMENTS PAR POSITION** sélectionne le ou les élément(s) dont vous avez passé la position dans *positionElém* et, facultativement, dans *tabPositions*, à l'intérieur de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans *liste*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste, vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste, la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration déployée/contractée.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande **SELECTIONNER ELEMENTS PAR POSITION** s'appliquera au premier objet dont le nom correspond.

La position des éléments est toujours exprimée en tenant compte de l'état déployé/contracté de la liste et de ses sous-listes. Passez des positions comprises entre 1 et la valeur retournée par **Nombre elements**. Si vous passez une valeur située en-dehors de cet intervalle, aucun élément n'est sélectionné.

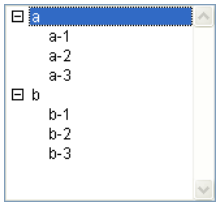
Si vous ne passez pas le paramètre *tabPositions*, le paramètre *positionElém* représente la position de l'élément à sélectionner.

Le paramètre facultatif *tabPositions* permet de sélectionner simultanément plusieurs éléments au sein de la *liste*. Vous devez passer dans *tabPositions* un tableau dont chaque ligne indique la position d'un élément à sélectionner.
Lorsque vous passez ce paramètre, l'élément désigné par le paramètre *positionElém* désigne parmi la sélection résultante le nouvel élément courant de la liste. Il peut appartenir ou non à l'ensemble d'éléments définis par le tableau. L'élément courant est notamment celui qui passe en mode édition si la commande **EDITER ELEMENT** est utilisée.

Note : Pour que plusieurs éléments puissent être sélectionnés simultanément dans une liste hiérarchique (manuellement ou par programmation), la propriété *multiSélection* doit avoir été activée pour cette liste. Cette propriété est définie via la commande **CHANGER PROPRIETES LISTE**.

Exemple

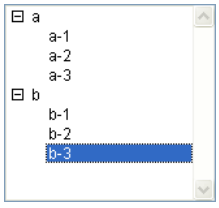
Soit une liste hiérarchique nommée *hList* affichée en mode Application :



Après l'exécution des lignes de code suivantes :

```
SELECTIONNER ELEMENTS PAR POSITION(hList;Nombre elements(hList))
```

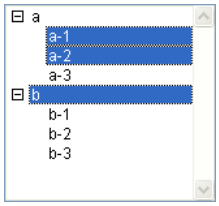
... le dernier élément visible est sélectionné :



Après l'exécution des lignes de code suivantes :

```
CHANGER PROPRIETES LISTE(hList;0;0;18;0;1)
 `Il est impératif de passer 1 en dernier paramètre pour autoriser les multi-sélections
TABLEAU ENTIER LONG($tab;3)
$tab{1}:=2
$tab{2}:=3
$tab{3}:=5
SELECTIONNER ELEMENTS PAR POSITION(hList;3;$tab)
 `Le 3e élément est désigné comme élément courant
```

... les 2e, 3e et 5e éléments de la liste hiérarchique sont sélectionnés :


a
a-1
a-2
a-3
b
b-1
b-2
b-3

SELECTIONNER ELEMENTS PAR REFERENCE

SELECTIONNER ELEMENTS PAR REFERENCE (liste ; réfElément {; tabRéfs})

Paramètre	Type		Description
liste	RefListe	➡	Numéro de référence de liste
réfElément	Entier long	➡	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste
tabRéfs	Tableau entier long	➡	Tableau de numéros de référence d'éléments

Description

La commande **SELECTIONNER ELEMENTS PAR REFERENCE** sélectionne le ou les élément(s) dont vous avez passé le numéro de référence dans *réfElément* et, facultativement, dans *tabRéfs*, parmi la liste dont vous avez passé la référence dans *liste*.

Si un élément n'est pas visible (car il est par exemple inclus dans une liste contractée), **SELECTIONNER ELEMENTS PAR REFERENCE** déploie la ou les sous-liste(s) correspondante(s) de manière à ce qu'il devienne visible.

Si vous ne passez pas le paramètre *tabRéfs*, le paramètre *réfElément* représente la référence de l'élément à sélectionner. Si le numéro d'élément ne correspond à aucun élément de la liste, la commande ne fait rien. Vous pouvez également passer la valeur 0 dans ce paramètre afin de désigner le dernier élément ajouté à la liste.

Le paramètre facultatif *tabRéfs* permet de sélectionner simultanément plusieurs éléments au sein de la liste. Vous devez passer dans *tabRéfs* un tableau dont chaque ligne indique la référence absolue d'un élément à sélectionner.
Dans ce cas, l'élément désigné par le paramètre *refElém* désigne parmi la sélection résultante le nouvel élément courant de la liste. Il peut appartenir ou non à l'ensemble d'éléments définis par le tableau. L'élément courant est notamment celui qui passe en mode édition si la commande **EDITER ELEMENT** est utilisée.

Note : Pour que plusieurs éléments puissent être sélectionnés simultanément dans une liste hiérarchique (manuellement ou par programmation), la propriété *multiSélection* doit avoir été activée pour cette liste. Cette propriété est définie via la commande **CHANGER PROPRIETES LISTE**.

Lorsque vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la description de la commande **AJOUTER A LISTE**.

Exemple

En supposant que *hList* est une liste dont les éléments ont des numéros de référence uniques, la méthode objet de bouton suivante sélectionne l'élément parent (s'il existe) de l'élément actuellement sélectionné :

```
$vlElémPos:=Elements selectionnes(hList) ` Récupérer la position de l'élément sélectionné
INFORMATION ELEMENT(hList;$vlElémPos;$vlElémRef;$vsElémText) ` Numéro de référence de cet élément
$vlParentElémRef:=Element parent(hList;$vlElémRef) ` Numéro de l'élément parent (s'il existe)
Si($vlParentElémRef>0)
  ` Sélection de l'élément parent
   SELECTIONNER ELEMENTS PAR REFERENCE(hList;Element parent(hList;$vlElémRef))
Fin de si
```

STOCKER LISTE

STOCKER LISTE (liste ; nomListe)

Paramètre	Type		Description
liste	RefListe	⇒	Numéro de référence de liste
nomListe	Chaîne	⇒	Nom de la liste tel qu'il doit apparaître dans l'éditeur d'énumérations en mode Développement

Description

La commande **STOCKER LISTE** sauvegarde la liste dont vous avez passé le numéro de référence dans *liste*, sous le nom que vous avez passé dans *nomListe*. La liste est stockée en tant qu'énumération dans l'éditeur d'énumérations du mode Développement.

Si une énumération de même nom existe déjà, son contenu est remplacé.

SUPPRIMER DANS LISTE

SUPPRIMER DANS LISTE ({* ;} liste ; réfElément | * {; *})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RefListe, Chaîne	⇒	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long, Opérateur	⇒	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément de la liste actuellement sélectionné
*		⇒	Si spécifié, effacer les sous-listes de la mémoire (le cas échéant) Si omis, ne pas effacer les sous-listes

Description

La commande **SUPPRIMER DANS LISTE** supprime l'élément désigné par le paramètre *réfElément* de la liste dont le numéro de référence ou le nom d'objet est passé dans *liste*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *liste* est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *liste* est une référence de liste hiérarchique (*RéfListe*). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Si vous passez * dans *réfElément*, vous supprimez l'élément actuellement sélectionné de la liste. Vous pouvez également passer 0 dans ce paramètre afin de demander la suppression du dernier élément ajouté à la liste.
Sinon, vous spécifiez le numéro de référence de l'élément à supprimer. Si le numéro ne correspond à aucun élément de la *liste*, la commande ne fait rien.

Si vous travaillez avec les numéros de référence des éléments, veillez à construire une liste dans laquelle les éléments ont des numéros de référence uniques, sinon vous ne pourrez les différencier. Pour plus d'informations sur ce point, reportez-vous à la description de la commande **AJOUTER A LISTE**.

Quel que soit l'élément que vous supprimez, vous pouvez passer un troisième paramètre optionnel, *, pour indiquer à 4D de supprimer automatiquement de la mémoire la sous-liste rattachée à l'élément, s'il en existe. Si vous ne passez pas ce paramètre, il est préférable de récupérer au préalable le numéro de référence de la sous-liste (éventuelle) rattachée à l'élément, de manière à pouvoir si besoin est supprimer cette sous-liste à l'aide de la commande **SUPPRIMER LISTE**.

Exemple

L'exemple suivant supprime l'élément sélectionné de la liste *hList*. Si une sous-liste est rattachée à l'élément, elle est supprimée (ainsi que toute sous-sous-liste) :

```
SUPPRIMER DANS LISTE(hList;*;*)
```

SUPPRIMER LISTE

SUPPRIMER LISTE (liste {; *})

Paramètre	Type		Description
liste	RefListe	⇒	Numéro de référence de liste
*		⇒	Si spécifié, effacer les sous-listes de la mémoire (s'il existe des sous-listes) Si omis, ne pas effacer les sous-listes

Description

La commande **SUPPRIMER LISTE** efface de la mémoire la liste hiérarchique dont vous avez passé le numéro de référence dans le paramètre *liste*.

Généralement, vous devez passer le paramètre optionnel *, afin que les sous-listes et les sous-éléments (s'il y en a) rattachés à la liste soient également effacés.

Il n'est pas nécessaire de supprimer une liste associée à un objet de formulaire via la Liste des propriétés : 4D charge et efface la liste automatiquement. Sinon, à chaque fois que vous chargez, copiez, extrayez d'un BLOB ou créez une liste par programmation, appelez la commande **SUPPRIMER LISTE** lorsque vous n'en avez plus besoin.

Si vous voulez supprimer une sous-liste rattachée à un élément (à tout niveau) d'une autre liste affichée dans un formulaire, procédez de la manière suivante :
1. Appelez **INFORMATION ELEMENT** avec l'élément parent pour obtenir le numéro de référence de la sous-liste.
2. Appelez **CHANGER ELEMENT** avec l'élément parent pour dissocier la sous-liste de l'élément de liste avant de l'effacer.
3. Appelez **SUPPRIMER LISTE** pour effacer la sous-liste dont vous avez obtenu le numéro de référence à l'aide de **INFORMATION ELEMENT**.

Exemple 1

Vous disposez, dans votre application, d'une routine de "nettoyage" chargée d'effacer tous les objets et données dont vous n'avez plus besoin lorsque, par exemple, une fenêtre ou un formulaire est refermé(e). A un endroit de cette routine, vous supprimez une liste hiérarchique qui peut avoir déjà été supprimée, suivant les actions de l'utilisateur dans le formulaire. Vous utilisez la fonction **Liste existante** pour effacer la liste uniquement si c'est nécessaire :

```
   ` Extrait de la sous-routine de nettoyage
Si(Liste existante(hlList))
   SUPPRIMER LISTE(hlList;*)
Fin de si
```

Exemple 2

Reportez-vous à l'exemple de la fonction **Charger liste**.

Exemple 3

Reportez-vous à l'exemple de la fonction **BLOB vers liste**.

TRIER LISTE

TRIER LISTE (liste {; > ou <})

Paramètre	Type		Description
liste	RefListe	⇒	Numéro de référence de liste
> ou <	Opérateur	⇒	Ordre de tri : > pour trier la liste dans l'ordre croissant ou < pour trier la liste dans l'ordre décroissant

Description

La commande **TRIER LISTE** effectue un tri sur la liste dont vous avez passé le numéro de référence dans le paramètre *liste*.

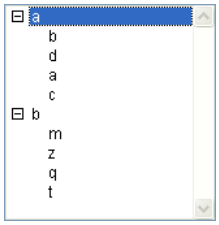
Pour effectuer un tri dans l'ordre croissant, passez > comme deuxième paramètre. Pour effectuer un tri dans l'ordre décroissant, passez < comme deuxième paramètre. Si vous omettez ce paramètre, **TRIER LISTE** effectue par défaut un tri croissant.

TRIER LISTE trie tous les niveaux de la liste : les éléments de la liste, puis les sous-éléments de chaque sous-liste, puis des sous-listes suivantes, etc., sont triés. C'est pourquoi généralement vous utiliserez la commande **TRIER LISTE** avec une liste affichée dans un formulaire. Le tri d'une sous-liste a moins d'intérêt car son ordre sera modifié dès qu'un appel à une liste se produira à un niveau supérieur.

TRIER LISTE ne modifie pas l'état courant déployé/contracté de la liste et de ses éventuelles sous-listes, ni l'élément courant. Cependant, comme l'élément courant peut être déplacé à la suite du tri, **Elements selectionnes** peut retourner une position différente avant et après le tri.

Exemple

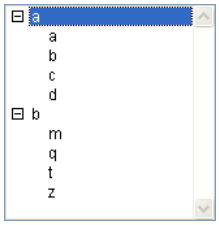
Voici la liste nommée *hList*, affichée ici en mode Application :



Après l'exécution du code suivant :

```
` Trier la liste et ses sous-listes dans l'ordre croissant
TRIER LISTE(hList;>)
```

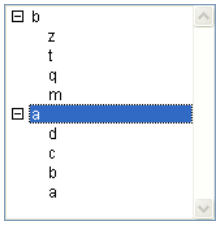
... la liste apparaît ainsi :



Après l'exécution du code suivant :

```
` Trier la liste et ses sous-listes dans l'ordre décroissant
TRIER LISTE(hList;<)
```

... la liste apparaît ainsi :



_o_REDESSINER LISTE

_o_REDESSINER LISTE (liste)

Paramètre	Type		Description
liste	RefListe	⇨	Numéro de référence de liste

Note de compatibilité

Cette commande est inutile depuis la version 11 de 4D, toutes les représentations des listes hiérarchiques sont désormais automatiquement redessinées. Lorsqu'elle est appelée, cette commande ne fait rien.

Menus

- Gestion des menus
- ACTIVER LIGNE MENU
- AJOUTER LIGNE MENU
- Creer menu
- EFFACER MENU
- FIXER BARRE MENUS
- FIXER ICONE LIGNE MENU
- FIXER MARQUE LIGNE MENU
- FIXER METHODE LIGNE MENU
- FIXER PARAMETRE LIGNE MENU
- FIXER PROPRIETE LIGNE MENU
- FIXER RACCOURCI LIGNE MENU
- FIXER STYLE LIGNE MENU
- FIXER TEXTE LIGNE MENU
- INACTIVER LIGNE MENU
- INSERER LIGNE MENU
- LIRE ICONE LIGNE MENU
- LIRE LIGNES MENU
- Lire marque ligne menu
- Lire methode ligne menu
- Lire modificateurs ligne menu
- Lire parametre ligne menu
- Lire parametre ligne menu selectionnee
- LIRE PROPRIETE LIGNE MENU
- Lire reference barre menu
- Lire style ligne menu
- Lire texte ligne menu
- Lire titre menu
- Lire touche ligne menu
- Menu choisi
- Nombre de lignes de menu
- Nombre de menus
- SUPPRIMER LIGNE MENU

Gestion des menus

Terminologie : La documentation sur les commandes de menus emploie indifféremment **commande de menu** et **ligne de menu** lorsqu'elle évoque une ligne d'un menu.

RefMenu et numéros de menus

Le langage de 4D propose deux modes de manipulation des menus et des barres de menus : par des références ou des numéros.

- La gestion de menus par référence (RefMenu) est le nouveau mode de gestion des menus, introduit depuis la version 11 de 4D. Ce mode donne accès à des fonctions avancées telles que la création d'interfaces entièrement dynamiques (menus créés "à la volée" sans devoir exister dans l'éditeur de menus) et la gestion de sous-menus hiérarchiques multi-niveaux.
- La gestion de menus et de barres de menus par numéro s'appuie sur les menus créés dans l'éditeur de menus en mode Développement. Chaque barre de menus et menu se voit attribuer un numéro fixe (correspondant à sa position dans l'éditeur). Ce numéro est utilisé par les commandes du langage pour désigner la barre ou le menu. La portée des commandes du langage appliquées aux menus gérés par numéro est la barre de menus courante.
  Ce fonctionnement correspond aux versions précédentes de 4D et obéit à plusieurs règles (décrites ci-dessous dans le paragraphe "Manipulation des menus par numéros"). Il peut toujours être utilisé mais ne permet pas de tirer parti des nouvelles fonctions proposées à partir de la version 11, notamment la gestion dynamique des menus et l'utilisation de sous-menus hiérarchiques : il n'est pas possible d'accéder à un sous-menu hiérarchique par un numéro.

Les deux modes de gestion des menus sont compatibles et peuvent être utilisés simultanément dans vos interfaces. La plupart des commandes du thème “Menus” acceptent indifféremment des numéros ou des références de menus.
Toutefois, la gestion par référence est conseillée car elle offre davantage de possibilités. A noter que si votre interface de menus est définie partiellement ou en totalité via l'éditeur de menus, il est parfaitement possible de l'exploiter sous forme de références à l'aide des commandes **Lire reference barre menu** et **LIRE LIGNES MENU**.

Manipulation des menus par référence

Lorsque les menus sont manipulés par l'intermédiaire de références RefMenu, il n’y a pas de différence de nature entre un menu et une barre de menus. Il s’agit dans les deux cas de listes de libellés. Seul leur usage diffère. Dans le cas d’une barre, chaque libellé correspond à un menu, composé de libellés. C’est également sur ce principe que repose la définition de menus hiérarchiques : chaque libellé peut à son tour être un menu, et ainsi de suite.

Lorsqu’un menu est géré par référence, toute modification effectuée sur ce menu durant la session est immédiatement répercutée à l’ensemble des instances de ce menu et ce, dans tous les process de la base.

RefMenu

A l’image des listes hiérarchiques, tous les menus disposent d'une référence unique, grâce à laquelle il pourront être identifiés durant toute la session. Cette référence, nommée par convention *RefMenu*, est un alphanumérique de 16 caractères. Toutes les commandes du thème “Menus” acceptent soit cette référence, soit un numéro de menu pour désigner un menu ou une barre.
Les références des menus peuvent être obtenues par les commandes **Creer menu**, **Lire reference barre menu** ou **LIRE LIGNES MENU**.

Manipulation des menus par numéro

Barres de menus

Les barres de menus peuvent être définies dans l'éditeur de menus en mode Développement. En mode de gestion par numéro, chaque barre de menus est identifiée par un numéro et par un nom. La première barre de menus (automatiquement créée par 4D) porte le numéro 1 et est nommée par défaut "Barre n°1". Vous pouvez la renommer dans l'éditeur de menu. Le nom d’une barre de menus peut contenir jusqu’à 31 caractères et doit être unique.
La barre n°1 est aussi la barre de menus utilisée par défaut. Si vous souhaitez ouvrir une application avec une barre de menus autre que la barre n°1, vous devez appeler la commande **FIXER BARRE MENUS** dans la **Méthode base Sur ouverture**. Il n'est pas possible de modifier par programmation le contenu même d'une barre de menus, en revanche les menus qui la composent peuvent être modifiés. La portée des commandes du langage appliquées aux menus statiques est la barre de menus courante. A chaque appel de la commande **FIXER BARRE MENUS** (sans le paramètre *), tous les menus et les commandes de menus retrouvent leur état originel tel que défini dans l'éditeur de menus.

Chaque barre de menus comporte par défaut trois menus — **Fichier**, **Edition** et **Mode**.

- Le menu **Fichier** ne contient qu'une commande de menu : **Quitter**. L'action standard Quitter lui est associée. Cette action affiche une boîte de dialogue de confirmation “Etes-vous certain ?” puis quitte l’application 4D en cas de validation. Dans le cas contraire, l’opération est annulée.

Note : Sous Mac OS, la commande de menu créé associée à l’action Quitter est automatiquement placée dans le menu de l’application, lorsque la base est exécutée sur ce système.

Vous pouvez renommer le menu Fichier, lui ajouter des commandes de menu ou le garder tel quel. Il est recommandé de toujours garder la commande de menu Quitter comme dernière commande du menu Fichier.

- Le menu **Edition** contient les commandes de menu d'édition standard. A chaque commande de ce menu est associée une action standard (Annuler, Couper, Copier, etc.). Vous pouvez ajouter des commandes à ce menu ou utiliser vos propres méthodes de gestion des actions d'édition.
- Le menu **Mode** contient par défaut la commande Retour au mode Développement. Cette commande permet de retourner au mode Développement (lorsqu'il est disponible) à partir du mode Application.

Note : 4D gère automatiquement les menus système **Aide** et **application** (Mac OS). Ces menus ne peuvent pas être modifiés, hormis pour la commande **A propos de 4D...,** qui peut être gérée à l'aide de la commande **APPELER SUR A PROPOS**.

Important : Les barres de menus sont "interprocess". Toute modification effectuée sur une barre en mode Développement sera répercutée dans tous les process où la barre est utilisée.

Numéros des menus et des commandes de menus

Comme les barres de menus, les menus sont numérotés. Le menu Fichier est généralement le menu 1. Les autres menus sont numérotés séquentiellement

de gauche à droite (2, 3, 4, etc.). Le menu **Application** (Mac OS) est exclu de cette numérotation. Sur toutes les plates-formes, le menu **Aide** est également exclu. Il est à noter que la commande **Nombre de menus** ne tient pas compte de ces menus. Si, par exemple, votre barre de menus est constituée des menus Fichier, Edition, Clients, Factures et Aide, **Nombre de menus** retournera 4 (en ignorant les menus système maintenus par 4D).
La numérotation des menus est importante lorsque vous travaillez, par exemple, avec la fonction **Menu choisi**.

Lorsqu'un menu est associé à un formulaire, le principe de numérotation est différent. Le premier menu ajouté commence avec le numéro 2049. Pour référencer un menu associé à un formulaire, ajoutez 2048 au numéro initial du menu.

Les commandes de chacun des menus sont numérotées séquentiellement de haut en bas, y compris les séparateurs. La commande supérieure a le numéro 1.

Barres de menus associées

Vous pouvez associer une barre de menus à un formulaire dans les Propriétés du formulaire (page Général). Ce type de barre est appelé "barre de menus de formulaire" dans cette section.
Les menus d'une barre de menus de formulaire sont ajoutés à la barre de menus courante lorsque le formulaire est affiché comme formulaire de sortie dans le mode Application.
Les barres de menus de formulaires sont référencées par un numéro et un nom. Si le numéro ou le nom de la barre de menus affichée avec le formulaire courant est le même que celui de la barre de menus associée au formulaire, cette dernière ne s'affiche pas.
Par défaut, lorsqu'un formulaire est affiché avec une barre de menus personnalisée, les commandes de la barre de menus courante sont inactivées, c'est-à-dire que leur sélection est sans effet. Vous pouvez modifier ce fonctionnement en cochant l'option **Barre de menus active** dans les Propriétés du formulaire : dans ce cas, les commandes de la barre de menus courante restent utilisables.
Dans tous les cas, la sélection d'une commande de menu provoque l'envoi d'un événement Sur menu sélectionné à la méthode formulaire ; vous pouvez alors utiliser la commande **Menu choisi** pour tester le menu sélectionné.

Menus rattachés

Les menus peuvent être rattachés à des barres de menus. Si un menu rattaché est modifié à l'aide d'une de ces commandes, chacune des instances de ce menu reflètera ces modifications. Pour plus d'informations sur ce point, reportez-vous au manuel Mode Développement de 4D.

Actions standard et méthodes associées aux commandes de menus

Chaque commande de menu peut être associée à une méthode projet ou une action standard. Si vous n'affectez pas de méthode ni d'action standard à une commande de menu, la sélection de cette commande de menu provoque la sortie du mode Application et le retour en mode Développement. Si seul le mode Application est disponible ou si l'utilisateur ne dispose pas des privilèges d'accès pour le mode Développement, cela provoquera la fermeture de l'application.
Les actions standard permettent d'effectuer diverses opérations courantes liées aux fonctions système (copier, quitter, etc.) ou de base de données 4D (ajouter enregistrement, tout sélectionner, etc.).

Vous pouvez associer à la fois une action standard et une méthode projet à une commande de menu. Dans ce cas, l'action standard n'est jamais exécutée ; toutefois, 4D utilise cette action pour activer/inactiver la commande de menu en fonction du contexte et lui associer éventuellement un comportement spécifique en fonction de la plate-forme (par exemple, l'action Préférences est passée dans le menu application sous Mac OS). Lorsqu'une commande de menu est inactivée, la méthode projet associée ne peut être exécutée.

ligneMenu=-1

Afin de faciliter la manipulation des lignes de menus, 4D propose un raccourci permettant de désigner la dernière ligne ajoutée au menu : il suffit pour cela de passer -1 dans le paramètre *ligneMenu*.
Ce principe est utilisable dans toutes les commandes du thème "Menus" manipulant des lignes de menus.

ACTIVER LIGNE MENU

ACTIVER LIGNE MENU (menu ; ligneMenu {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Entier long	⇒	Numéro de référence du process

Description

ACTIVER LIGNE MENU active la commande de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans *menu* et *ligneMenu*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si le paramètre *ligneMenu* désigne un sous-menu hiérarchique, toutes les lignes de ce menu et de ses éventuels sous-menus sont activées. Cette commande fonctionne également avec une barre de menus créée avec la commande **Creer menu** et installée avec la commande **FIXER BARRE MENUS**.

Si vous omettez le paramètre *process*, **ACTIVER LIGNE MENU** s'applique à la barre de menus du process courant. Sinon, **ACTIVER LIGNE MENU** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Astuce : Pour activer/inactiver toutes les lignes de menus en une fois, passez 0 (zéro) dans *ligneMenu*.

AJOUTER LIGNE MENU

AJOUTER LIGNE MENU (menu ; libelléLigne {; sousMenu {; process {; *}}})

Paramètre	Type		Description
menu	Entier long, RefMenu	➡	Numéro de menu ou Référence de menu
libelléLigne	Texte	➡	Libellé du ou des nouvelle(s) ligne(s) de menu
sousMenu	RefMenu	➡	Référence du sous-menu associé à la ligne
process	Entier long	➡	Numéro de référence du process
*	Opérateur	➡	Si passé : considérer les métacaractères comme des caractères standard

Description

La commande **AJOUTER LIGNE MENU** ajoute une ou plusieurs ligne(s) au menu dont vous avez passé le numéro ou la référence dans *menu.*

Si vous omettez le paramètre *process*, **AJOUTER LIGNE MENU** s'applique à la barre de menus du process courant. Sinon, **AJOUTER LIGNE MENU** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Si vous ne passez pas le paramètre *, **AJOUTER LIGNE MENU** vous permet d'ajouter une ou plusieurs lignes de menu en un seul appel. Vous définissez les lignes à ajouter à l'aide du paramètre *libelléLigne*, de la manière suivante :

- Chaque ligne est séparée des autres par un point-virgule ";", *"ligne1;ligne2;ligne3".*
- Pour inactiver une ligne, placez une parenthèse ouvrante "(" dans son libellé.
- Pour définir une ligne de séparation, passez la valeur "-" ou *"(-"* en tant que libellé.
- Pour définir le style de caractères d'une ligne, placez dans son libellé le symbole inférieur à "<" suivi d'une lettre. Voici les différents codes :

<B	Gras
<I	Italique
<U	Souligné

- Pour associer une coche à une ligne, insérez dans son libellé un point d'exclamation "!" suivi du caractère que vous voulez utiliser comme coche. Sous Mac OS, le caractère est affiché ; sous Windows, une coche standard est affichée (quel que soit le caractère passé).
- Pour associer une icône à une ligne, insérez dans son libellé un accent circonflexe "^" suivi d'un caractère dont le code moins 48 plus 256 (ou plus 208) représente un numéro de ressource d'icône Mac OS.
- Pour ajouter un raccourci clavier à une ligne, insérez dans son libellé une barre oblique "/" suivie du caractère de raccourci.

Note : Utilisez les menus avec un nombre "raisonnable" de lignes. Si, par exemple, vous voulez afficher plus de 50 lignes, envisagez plutôt d'employer une zone de défilement dans un formulaire.

Si vous passez le paramètre *, les caractères "spéciaux" inclus dans les libellés des lignes (*; (!*...) seront considérés comme des caractères standard et non comme des métacaractères. Ce principe vous permet de créer des lignes avec un libellé tel que "**Copier (spécial)...**" ou "**Chercher/Remplacer...**". A noter que lorsque le paramètre * est passé, vous ne pouvez pas créer plusieurs lignes en un seul appel, le caractère ";" étant considéré comme un caractère standard.

Note : Les commandes **LIRE LIGNES MENU** et **Lire texte ligne menu** retourneront ou non les métacaractères d'un libellé en fonction de son mode de création : s'il a été créé avec l'option *, les métacaractères seront retournés en tant que caractères standard.

Le paramètre facultatif *sousMenu* vous permet de désigner un menu comme ligne ajoutée et donc de définir un sous-menu hiérarchique. Vous devez passer dans ce paramètre une référence de menu (chaîne de type *RefMenu*) désignant un menu créé par exemple à l'aide de la commande **Creer menu**. Si la commande ajoute plusieurs lignes de menus, le sous-menu est associé à la première ligne.

Important : Les nouvelles lignes n'ont pas de méthodes ou d'actions associées. Vous devez leur associer une action ou une méthode via les commandes **FIXER PROPRIETE LIGNE MENU** ou **FIXER METHODE LIGNE MENU** ou encore les gérer à partir d'une méthode formulaire qui utilise la fonction **Menu choisi**.

Exemple

L'exemple suivant ajoute les noms des polices de caractères disponibles dans un menu **Polices** qui, dans cet exemple, est le sixième menu de la barre de menus courante :

```
   ` Dans la méthode base Sur ouverture
   ` La liste des polices est chargée et les libellés construits
LISTE DES POLICES(<>asPolicesDispo)
<>atPoliceCmdMenus:=""
Boucle($vlPolice;1;Taille tableau(<>asPolicesDispo))
   <>atPoliceCmdMenus:=<>atPoliceCmdMenus+";"+<>asPolicesDispo{$vlPolice}
Fin de boucle
```

Ensuite, dans toute méthode formulaire ou projet, vous pouvez écrire :

```
AJOUTER LIGNE MENU(6;<>atPoliceCmdMenus)
```

Creer menu

Creer menu {(menu)} -> Résultat

Paramètre	Type		Description
menu	RefMenu, Entier long, Chaîne	⇒	Référence de menu ou Numéro ou Nom de barre de menus
Résultat	RefMenu	⇐	Référence du menu

Description

La commande **Creer menu** permet de créer un nouveau menu en mémoire. Ce menu n'existera qu'en mémoire et ne sera pas ajouté dans l'éditeur de menus en mode Développement. Toute modification effectuée sur ce menu durant la session sera immédiatement répercutée à l'ensemble des instances de ce menu et ce, dans tous les process de la base.

La commande retourne un identifiant unique de type *RefMenu* pour le nouveau menu.

- Si vous ne passez pas le paramètre facultatif *menu*, le menu sera créé vide. Vous devrez le construire et le gérer à l'aide des commandes **EFFACER MENU**, **FIXER TEXTE LIGNE MENU**, etc.
- Si vous passez le paramètre *menu*, le menu créé sera une copie exacte du menu source désigné par ce paramètre. Toutes les propriétés du menu source, y compris les éventuels sous-menus associés, seront appliquées au nouveau menu. A noter qu'une nouvelle référence *RefMenu* est créée pour le menu source et pour chaque sous-menu associé existant.

Vous pouvez passer dans *menu* soit une référence de menu valide, soit un numéro ou un nom de barre de menus défini en mode Développement. Dans ce dernier cas, le nouveau menu sera constitué des menus et sous-menus de la barre d'origine.

Note : Si vous passez une valeur invalide dans *menu*, un menu vide est créé.

Un menu créé par cette commande peut être utilisé en tant que barre de menus à l'aide de la commande **FIXER BARRE MENUS**.

Lorsque vous n'avez plus besoin d'un menu créé par **Creer menu**, n'oubliez pas d'appeler la commande **EFFACER MENU** afin de libérer la mémoire qu'il occupe.

Exemple

Reportez-vous à l'exemple de la commande **FIXER BARRE MENUS**.

EFFACER MENU

EFFACER MENU (menu)

Paramètre	Type		Description
menu	RefMenu	➡	Référence de menu

Description

La commande **EFFACER MENU** efface de la mémoire le menu dont vous avez passé l'identifiant dans *menu*. Ce menu doit avoir été créé par la commande **Creer menu**. La règle est la suivante : à chaque **Creer menu** doit correspondre un **EFFACER MENU**.

La commande efface toutes les instances du *menu* dans toutes les barres de menus et tous les process. Si le menu appartient à une barre de menus en cours d'utilisation, il continuera à fonctionner mais ne pourra plus être modifié. Il ne sera réellement effacé de la mémoire que lorsque la dernière barre de menus dans laquelle il figure ne sera plus utilisée.

Cette commande peut être appliquée aux menus utilisés comme barres de menus.

Les sous-menus éventuellement utilisés par *menu* ne sont pas effacés s'ils ont été créés directement via la commande **Creer menu**. Vous devez dans ce cas les effacer individuellement (cf. règle énoncée ci-dessus). En revanche, s'ils sont issus de la duplication d'un menu existant, n'appelez pas **EFFACER MENU** avec leurs instances car 4D les efface automatiquement.

Exemple

Cet exemple illustre les cas d'utilisation de cette commande :

```
 nouvMenu:=Creer menu
 AJOUTER LIGNE MENU(nouvMenu;"Test1")
 sousMenu:=Creer menu
 AJOUTER LIGNE MENU(sousMenu;"SousTest1")
 AJOUTER LIGNE MENU(sousMenu;"SousTest2")  // Création de lignes dans le sous-menu

 AJOUTER LIGNE MENU(nouvMenu;"Test2";sousMenu)  // Attacher le sous-menu au menu

  //A ce moment, le sous-menu est attaché au menu et si on n'en a plus besoin par la suite, c'est le bon
emplacement pour l'effacer.
  //L'effacement ne supprime pas tout de suite sousMenu car il est encore utilisé par nouvMenu. sousMenu sera
supprimé quand nouvMenu le sera.
  //Effacer le sous-menu à cet instant permet d'économiser la mémoire
 EFFACER MENU(sousMenu)

 $result1:=Pop up menu dynamique(nouvMenu) //Utilisation du menu
 copieMenu:=Creer menu(nouvMenu) // Création d'un menu par copie de nouvMenu (et donc duplication de sousMenu)
 EFFACER MENU(nouvMenu) // On n'a plus besoin de nouvMenu.

 $result2:=Pop up menu dynamique(copieMenu)
 EFFACER MENU(copieMenu)
  //Il ne faut pas chercher à effacer le sous-menu de copieMenu puisqu'il n'a pas été créé directement par Creer
menu
  //La règle est respectée : à chaque Creer menu correspond un EFFACER MENU
```

FIXER BARRE MENUS

FIXER BARRE MENUS (barre {; process}{; *})

Paramètre	Type		Description
barre	Entier long, Chaîne, RefMenu	⇒	Numéro ou nom de la barre de menus ou Référence de menu
process	Entier long	⇒	Numéro de référence du process
*	Opérateur	⇒	Conserver l'état de la barre de menus

Description

La commande **FIXER BARRE MENUS** remplace la barre de menus courante par la barre de menus *barre*, pour le process en cours uniquement. Vous pouvez passer dans le paramètre *barre* soit le numéro soit le nom de la nouvelle barre. Vous pouvez également passer une référence unique de menu (type *RefMenu*, chaîne de 16 caractères). Lorsque vous travaillez avec des références, les menus peuvent être utilisés comme barres de menus et inversement (cf. section **Gestion des menus**).

Note : Le nom d'une barre de menus peut contenir jusqu'à 31 caractères et doit être unique.

Si vous passez le paramètre optionnel *process*, c'est la barre de menus du process spécifié qui sera remplacée par la *barre*.

Note : Si vous passez un paramètre *RefMenu* dans *barre*, le paramètre *process* est inutile et sera ignoré.

Le paramètre optionnel * vous permet de conserver l'état de la barre de menus. Si ce paramètre est omis, **FIXER BARRE MENUS** réinitialise la barre de menus lors de l'exécution de la commande.
Imaginez, par exemple, que l'instruction **FIXER BARRE MENUS(1)** soit exécutée. Ensuite, plusieurs commandes de menu sont désactivées à l'aide de la commande **INACTIVER LIGNE MENU**.
Si **FIXER BARRE MENUS(1)** est exécutée une seconde fois, soit à partir du même process, soit à partir d'un autre process, toutes les commandes de menu retournent à leur état d'activation initial.
Si **FIXER BARRE MENUS(1;*)** est exécutée, la barre de menus conservera son état précédent, les commandes de menu qui étaient inactivées le resteront.

Note : Si vous passez un paramètre *RefMenu* dans *barre*, le paramètre * est inutile et sera ignoré.

Lorsqu'un utilisateur arrive en mode Application, la première barre de menus s'affiche (Barre n° 1). Vous pouvez changer cette barre de menus par défaut en spécifiant la barre que vous voulez dans la **Méthode base Sur ouverture**, ou dans la méthode de démarrage associée à un utilisateur.

Exemple 1

L'exemple suivant remplace la barre de menus courante par la barre de menus n° 3 et initialise l'état des commandes des menus :

```
FIXER BARRE MENUS(3)
```

Exemple 2

L'exemple suivant remplace la barre de menus courante par la barre de menus nommée "BarreForm1" et conserve l'état des commandes des menus : celles qui étaient précédemment inactivées apparaîtront inactivées :

```
FIXER BARRE MENUS("BarreForm1";*)
```

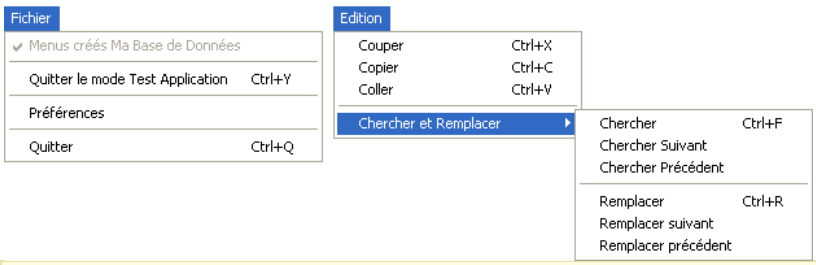
Exemple 3

L'exemple suivant remplace la barre de menus courante par la barre de menus n° 3 pendant que des enregistrements sont en cours de modification. Une fois les enregistrements modifiés, la barre de menus n° 2 est réaffichée. L'état des commandes de ce menu est conservé :

```
FIXER BARRE MENUS(3) ` Définir la barre de menus n° 3 pour le formulaire suivant
TOUT SELECTIONNER([Clients])
MODIFIER SELECTION([Clients]) ` Afficher la sélection
FIXER BARRE MENUS(2;*) ` Après modification, retour à la barre de menus n° 2
```

Exemple 4

Dans cet exemple complet, nous allons créer par programmation une barre comportant les menus Fichier et Edition suivants :




```
 `Méthode de création menu Fichier
C_TEXTE(FileMenu) ` FileMenu contiendra la référence du menu Fichier
FileMenu:=Creer menu
```

```
 INSERER LIGNE MENU(FileMenu;-1;Lire chaine dans liste(131;29)+" Ma Base de Données(")
 FIXER MARQUE LIGNE MENU(FileMenu;1;Caractere(18))
 INSERER LIGNE MENU(FileMenu;-1;"(-")
 INSERER LIGNE MENU(FileMenu;-1;"Quitter le mode Test Application/Y")
 FIXER PROPRIETE LIGNE MENU(FileMenu;3;Action standard associée;Retour au mode Développement)
 INSERER LIGNE MENU(FileMenu;-1;"(-")
 INSERER LIGNE MENU(FileMenu;-1;Lire chaine dans liste(131;26))
 FIXER PROPRIETE LIGNE MENU(FileMenu;6;Action standard associée;Action propriétés de la base) `Preferences
 INSERER LIGNE MENU(FileMenu;-1;"(-")
 INSERER LIGNE MENU(FileMenu;-1;Lire chaine dans liste(131;30))
 FIXER PROPRIETE LIGNE MENU(FileMenu;7;Action standard associée;Action quitter) `Quitter
 FIXER RACCOURCI LIGNE MENU(FileMenu;7;Code de caractere("Q"))

  `Méthode de création menu Chercher et Remplacer
 C_TEXTE(FindAndReplaceMenu) `FindAndReplaceMenu contiendra la référence du menu Chercher remplacer
 FindAndReplaceMenu:=Creer menu
 AJOUTER LIGNE MENU(FindAndReplaceMenu;"Chercher;Chercher Suivant;Chercher Précédent;(-;Remplacer;Remplacer
suivant;Remplacer précédent")
 FIXER RACCOURCI LIGNE MENU(FindAndReplaceMenu;1;Code de caractere("F"))
 FIXER RACCOURCI LIGNE MENU(FindAndReplaceMenu;5;Code de caractere("R"))
 FIXER METHODE LIGNE MENU(FindAndReplaceMenu;1;"MaMethodechercher")

  `Méthode de création menu Edition
 C_TEXTE(EditMenu) `EditMenu contiendra la référence du menu Edition
 EditMenu:=Creer menu
 AJOUTER LIGNE MENU(EditMenu;"Couper;Copier;Coller")
 FIXER RACCOURCI LIGNE MENU(EditMenu;1;Code de caractere("X"))
 FIXER PROPRIETE LIGNE MENU(EditMenu;1;Action standard associée;Action couper)
 FIXER RACCOURCI LIGNE MENU(EditMenu;2;Code de caractere("C"))
 FIXER PROPRIETE LIGNE MENU(EditMenu;2;Action standard associée;Action copier)
 FIXER RACCOURCI LIGNE MENU(EditMenu;3;Code de caractere("V"))
 FIXER PROPRIETE LIGNE MENU(EditMenu;3;Action standard associée;Action coller)
 INSERER LIGNE MENU(EditMenu;-1;"(-")
 INSERER LIGNE MENU(EditMenu;-1;"Chercher et Remplacer";FindAndReplaceMenu) ` ligne qui aura le sous menu

 main_Bar:=Creer menu ` Cree la barre constituée des autres menus
 INSERER LIGNE MENU(main_Bar;-1;Lire chaine dans liste(79;1);FileMenu)
 AJOUTER LIGNE MENU(main_Bar;"Edition";EditMenu)

 FIXER BARRE MENUS(main_Bar)
```

FIXER ICONE LIGNE MENU

FIXER ICONE LIGNE MENU (menu ; ligneMenu ; reflcône {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Référence de menu ou Numéro de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
reflcône	Texte, Entier long	⇒	Nom ou numéro de l'image à associer à la ligne de menu
process	Entier long	⇒	Numéro de process

Description

La commande **FIXER ICONE LIGNE MENU** permet de modifier l'icône associée à la ligne de menu désignée par les paramètres *menu* et *ligneMenu*.

Vous pouvez passer dans *menu* un identifiant unique de menu (*RefMenu*) ou un numéro de menu. Si vous passez un identifiant unique, la commande s'appliquera à toutes les instances du menu dans tous les process. Dans ce cas, le paramètre *process* est ignoré s'il est passé. Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif *process*.

Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au menu.

Passez dans le paramètre *reflcône* l'image devant être utilisée comme icône. Vous pouvez utiliser une image de la bibliothèque ou une référence d'image.

- Image de la bibliothèque : vous pouvez passer soit le nom soit le numéro de l' image. Il est généralement préférable d'utiliser le numéro plutôt que le nom, car les numéros d'images sont des identifiants uniques, ce qui n'est pas le cas des noms.
- Référence d'image : l'image doit se trouver dans le dossier **Resources** de la base et vous devez utiliser une syntaxe du type "file:{cheminaccès}nomFichier" dans *reflcône*. Pour plus d'informations sur le dossier **Resources**, reportez-vous à la section **Ressources**.

Exemple

Utilisation d'une image se trouvant dans le dossier Resources de la base :

```
FIXER ICONE LIGNE MENU($RefMenu;2;"File:French.lproj/spot.png")
```

FIXER MARQUE LIGNE MENU

FIXER MARQUE LIGNE MENU (menu ; ligneMenu ; marque {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
marque	Chaîne	⇒	Nouvelle marque de ligne de menu
process	Entier long	⇒	Numéro de référence du process

Description

La commande **FIXER MARQUE LIGNE MENU** remplace la marque (ou "coche") de la ligne de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans *menu* et *ligneMenu* par le premier caractère de la chaîne que vous avez passée dans *marque* (sous Mac OS) ou par la coche standard (sous Windows). Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si vous omettez le paramètre *process*, **FIXER MARQUE LIGNE MENU** s'applique à la barre de menus du process courant. Sinon, **FIXER MARQUE LIGNE MENU** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Si vous passez une chaîne vide dans *marque*, vous supprimez toute marque de la ligne de menu.
Sinon :

- Sous Mac OS, le premier caractère de la chaîne devient la marque de la ligne de menu (généralement, le **Caractere(18)**, qui est la coche standard de Mac OS, est utilisé).
- Sous Windows, la marque standard de Windows est associée au menu.

Exemple

Reportez-vous à l'exemple de la commande **Lire marque ligne menu**.

FIXER METHODE LIGNE MENU

FIXER METHODE LIGNE MENU (menu ; ligneMenu ; nomMéthode {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Référence de menu ou Numéro de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
nomMéthode	Chaîne	⇒	Nom de la méthode
process	Entier long	⇒	Numéro de process

Description

La commande **FIXER METHODE LIGNE MENU** permet de modifier la méthode projet 4D associée à la ligne de menu désignée par les paramètres *menu* et *ligneMenu*.
Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans *menu* un identifiant unique de menu (*RefMenu*) ou un numéro de menu. Si vous passez un identifiant unique, la commande s'appliquera à toutes les instances du menu dans tous les process. Dans ce cas, le paramètre *process* est ignoré s'il est passé. Si vous passez un numéro de menu, la commande s'appliquera au menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif *process*.

Passez dans *nomMéthode* le nom de la méthode 4D sous la forme d'une chaîne de caractères (expression).

Note : Si la ligne de menu correspond au libellé d'un sous-menu hiérarchique, la méthode ne sera pas appelée lorsque la ligne de menu sera sélectionnée.

Exemple

Reportez-vous aux exemple de la commande **FIXER BARRE MENUS**.

FIXER PARAMETRE LIGNE MENU

FIXER PARAMETRE LIGNE MENU (menu ; ligneMenu ; param)

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Référence de menu ou Numéro de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
param	Chaîne	⇒	Chaîne à associer en tant que paramètre

Description

La commande **FIXER PARAMETRE LIGNE MENU** vous permet d'associer une chaîne de caractères personnalisée à la ligne de menu désignée par les paramètres *menu* et *ligneMenu*.
Ce paramètre sera principalement utilisé par la commande **Pop up menu dynamique**.

Exemple

Ce code permet de proposer un menu comportant le libellé des fenêtres ouvertes et de récupérer le numéro de la fenêtre choisie :

```
LISTE FENETRES($alFenetre)
$tRefMenu:=Creer menu
Boucle($i;1;Taille tableau($alFenetre))
   AJOUTER LIGNE MENU($tRefMenu;Titre fenetre($alFenetre{$i}))  //Libellé de la ligne du menu
   FIXER PARAMETRE LIGNE MENU($tRefMenu;-1;Chaine($alFenetre{$i}))  //Valeur retournée par la ligne du menu
Fin de boucle
$tRefFenetre:=Pop up menu dynamique($tRefMenu)
EFFACER MENU($tRefMenu)
```

FIXER PROPRIETE LIGNE MENU

FIXER PROPRIETE LIGNE MENU (menu ; ligneMenu ; propriété ; valeur {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Référence de menu ou Numéro de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
propriété	Chaîne	⇒	Type de propriété
valeur	Expression	⇒	Valeur de la propriété
process	Entier long	⇒	Numéro de process

Description

La commande **FIXER PROPRIETE LIGNE MENU** permet de fixer la *valeur* de la *propriété* pour la ligne de menu désignée par les paramètres *menu* et *ligneMenu*.
Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans *menu* un identifiant unique de menu (*RefMenu*) ou un numéro de menu. Si vous passez un identifiant unique, la commande s'appliquera à toutes les instances du menu dans tous les process. Dans ce cas, le paramètre *process* est ignoré s'il est passé. Si vous passez un numéro de menu, la commande s'appliquera au menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif *process*.

Passez dans le paramètre *propriété* la propriété dont vous souhaitez modifier la valeur et dans *valeur*, la nouvelle valeur. Pour le paramètre *propriété*, vous pouvez utiliser l'une des constantes du thème "**Propriétés des lignes de menu**" ou toute valeur personnalisée :
Propriété standard : les constantes du thème "**Propriétés des lignes de menu**" ainsi que leurs valeurs possibles sont décrites ci-dessous. A noter que dans le cas de la propriété Action standard associée, vous pouvez passer une des constantes du thème "**Valeurs pour Actions standard associée**" dans le paramètre *valeur*.

Constante	Type	Valeur	Comment
Action standard associée	Chaîne	4D_standard_action	Associer une action standard à la ligne de menu Voir les constantes du thème "**Valeurs pour Actions standard associée**"
Autorisations d'accès	Chaîne	4D_access_group	Affecter un groupe d'accès à la commande 0 = Sans restriction >0 = Numéro de groupe
Démarrer un process	Chaîne	4D_start_new_process	Activer l'option "Démarrer un nouveau process" 0 = Non, 1 = Oui

Voici les constantes du thème "**Valeurs pour Actions standard associée**" :

Constante	Type	Valeur
Action afficher Presse papiers	Entier long	23
Action ajouter sous enreg	Entier long	14
Action annuler	Entier long	17
Action coller	Entier long	20
Action copier	Entier long	19
Action couper	Entier long	18
Action CSM	Entier long	36
Action dernier enregistrement	Entier long	6
Action dernière page	Entier long	11
Action effacer	Entier long	21
Action enregistrement précédent	Entier long	4
Action enregistrement suivant	Entier long	3
Action modifier sous enreg	Entier long	12
Action ne pas valider	Entier long	1
Action page précédente	Entier long	9
Action page suivante	Entier long	8
Action premier enregistrement	Entier long	5
Action première page	Entier long	10
Action propriétés de la base	Entier long	32
Action quitter	Entier long	27
Action répéter	Entier long	31
Action supprimer enregistrement	Entier long	7
Action supprimer sous enreg	Entier long	13
Action test application	Entier long	26
Action tout sélectionner	Entier long	22
Action valider	Entier long	2
Pas d'action	Entier long	0
Retour au mode Développement	Entier long	35

Pour plus d'informations sur les propriétés standard des lignes de menus, reportez-vous au chapitre "Créer des menus personnalisés" dans le manuel Mode Développement.

Propriété personnalisée : vous pouvez passer dans *propriété* tout texte personnalisé et lui associer une *valeur* de type texte, numérique ou booléen. Cette *valeur* sera stockée avec l'élément et pourra être récupérée via la commande **LIRE PROPRIETE LIGNE MENU**. Vous pouvez utiliser toute chaîne personnalisée dans le paramètre *propriété*, veillez simplement à ne pas utiliser de libellé utilisé par 4D (par convention, les propriétés définies par 4D débutent par les caractères "4D_").

Note : Si la ligne de menu correspond au libellé d'un sous-menu hiérarchique, l'action standard ne sera pas appelée lorsque la ligne de menu sera

sélectionnée.

FIXER RACCOURCI LIGNE MENU

FIXER RACCOURCI LIGNE MENU (menu ; ligneMenu ; touche ; modificateurs {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro du menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
touche	Chaîne, Entier long	⇒	Lettre du raccourci clavier ou code de caractère du raccourci clavier (ancienne syntaxe)
modificateurs	Entier long	⇒	Modificateur(s) à associer au raccourci (ignoré si un code de touche est passé)
process	Entier long	⇒	Numéro de référence du process

Description

La commande **FIXER RACCOURCI LIGNE MENU** remplace la touche du raccourci clavier associé à la ligne de menu désignée par *menu* et *ligneMenu*, par le caractère dont vous avez passé le code de caractère ou le texte dans *touche*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*. La touche définie sera combinée à la touche **Ctrl** (Windows) ou **Commande** (Macintosh) pour définir le nouveau raccourci clavier.

Passez dans le paramètre *touche* la lettre désignant la touche de raccourci, par exemple “U” pour définir le raccourci **Ctrl+U** (Windows) ou **Commande+U** (Mac OS).

Le paramètre *modificateurs* vous permet d’associer un ou plusieurs modificateur(s) additionnel(s) au raccourci standard. Vous pouvez ainsi définir des raccourcis du type **Ctrl+Alt+Maj+Z** (Windows) ou **Cmd+Option+Maj+Z** (Mac OS). Vous pouvez passer dans *modificateurs* les valeurs suivantes :

- 256 pour la touche **Commande** (Mac OS) ou **Ctrl** (Windows)
- 512 pour la touche **Majuscule**
- 2048 pour la touche **Option** (Mac OS) ou **Alt** (Windows)
- Pour associer plusieurs touches, cumulez leurs valeurs.

Note : Vous pouvez définir les valeurs à passer à l’aide des constantes Masque touche commande , Masque touche majuscule et Masque touche option du thème **Evénements (Modifiers)**.

La touche **Ctrl** (Windows) ou **Commande** (Mac OS) est automatiquement ajoutée par 4D au raccourci clavier, que vous l'ayez explicitement indiquée ou non dans *modificateurs*. Il n'est donc pas nécessaire d'ajouter la valeur 256 à ce paramètre, sauf si cette touche est le seul modificateur, auquel cas vous devez passer la valeur 256 ou la constante correspondante dans *modificateurs*.

Note : Par compatibilité, la commande admet également un code de caractère comme paramètre *touche* (ancienne syntaxe). Dans ce cas, le paramètre *modificateurs* n’est pas pris en compte et il peut être omis. Le raccourci sera uniquement associé au modificateur **Ctrl** (Windows) ou **Commande** (Mac OS).

Si vous ne passez pas le paramètre *process*, **FIXER RACCOURCI LIGNE MENU** est appliquée à la barre de menus du process courant. Sinon, **FIXER RACCOURCI LIGNE MENU** est appliquée à la barre de menus du process dont la référence est passée dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Si vous passez *0* (zéro) dans *touche*, l'équivalent clavier de la commande de menu est supprimé.

Exemple 1

Définition du raccourci Ctrl+Maj+U (Windows) et Cmd+Maj+U (Mac OS) pour la ligne “Souligné” :

```
FIXER TEXTE LIGNE MENU(menuRef;1;"Souligné")
FIXER RACCOURCI LIGNE MENU(menuRef;1;"U";Masque touche majuscule)
```

Exemple 2

Définition du raccourci Ctrl+R (Windows) et Cmd+R (Mac OS) pour la ligne "Recommencer" :

```
INSERER LIGNE MENU(FileMenu;-1;"Recommencer")
FIXER RACCOURCI LIGNE MENU(FileMenu;-1;"R";Masque touche commande)
```

FIXER STYLE LIGNE MENU

FIXER STYLE LIGNE MENU (menu ; ligneMenu ; styleLigne {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
styleLigne	Entier long	⇒	Nouveau style de la ligne de menu
process	Entier long	⇒	Numéro de référence du process

Description

La commande **FIXER STYLE LIGNE MENU** remplace le style de police de la ligne de menu, dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans *menu* et *ligneMenu*, par le style de police que vous avez passé dans *styleLigne*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si vous omettez le paramètre *process*, **FIXER STYLE LIGNE MENU** s'applique à la barre de menus du process courant. Sinon, **FIXER STYLE LIGNE MENU** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Vous pouvez définir le style de l'élément dans le paramètre *styleLigne*. Vous passez une ou une combinaison des constantes prédéfinies suivantes, placées dans le thème **Styles de caractères** :

Constante	Type	Valeur
Gras	Entier long	1
Italique	Entier long	2
Normal	Entier long	0
Souligné	Entier long	4

FIXER TEXTE LIGNE MENU

FIXER TEXTE LIGNE MENU (menu ; ligneMenu ; libelléElément {; process}{; *})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
libelléElément	Chaîne	⇒	Nouveau libellé de la ligne de menu
process	Entier long	⇒	Numéro de référence de process
*	Opérateur	⇒	Si passé : considérer les métacaractères comme des caractères standard

Description

La commande **FIXER TEXTE LIGNE MENU** remplace le libellé de la ligne de menu, dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans *menu* et *ligneMenu*, par le libellé que vous avez passé dans *texteLigne*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si vous ne passez pas le paramètre *, les caractères "spéciaux" inclus dans *texteLigne* (tels que (; ou !) seront considérés comme des caractères d'instruction (métacaractères). Par exemple, pour définir une ligne de menu comme ligne de séparation, insérez le caractère “-” dans *texteLigne*. Si vous passez le paramètre *, les caractères "spéciaux" seront considérés comme des caractères standard. Reportez-vous à la description de la commande **AJOUTER LIGNE MENU** pour plus de détails sur ces caractères.

Si vous omettez le paramètre *process*, **FIXER TEXTE LIGNE MENU** s'applique à la barre de menus du process courant. Sinon, **FIXER TEXTE LIGNE MENU** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

INACTIVER LIGNE MENU

INACTIVER LIGNE MENU (menu ; ligneMenu {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Entier long	⇒	Numéro de référence du process

Description

INACTIVER LIGNE MENU désactive la commande de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans *menu* et *ligneMenu*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si le paramètre *ligneMenu* désigne un sous-menu hiérarchique, toutes les lignes de ce menu et de ses éventuels sous-menus sont inactivées. Cette commande fonctionne également avec une barre de menus créée avec la commande **Creer menu** et installée avec la commande **FIXER BARRE MENUS**.

Si vous omettez le paramètre *process*, **INACTIVER LIGNE MENU** s'applique à la barre de menus du process courant. Sinon, **INACTIVER LIGNE MENU** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Astuce : Pour activer/inactiver toutes les lignes de menus en une fois, passez 0 (zéro) dans *ligneMenu*.

INSERER LIGNE MENU

INSERER LIGNE MENU (menu ; aprèsLigne ; libelléElément {; sousMenu {; process}}{; *})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
aprèsLigne	Entier long	⇒	Numéro de commande de menu
libelléElément	Chaîne	⇒	Libellé de la ligne de menu à insérer
sousMenu	RefMenu	⇒	Référence du sous-menu associé à la ligne
process	Entier long	⇒	Numéro de référence de process
*	Opérateur	⇒	Si passé : considérer les métacaractères comme des caractères standard

Description

La commande **INSERER LIGNE MENU** insère de nouvelles lignes dans le menu dont vous avez passé le numéro ou la référence dans *menu* et les place après la ligne de menu dont le numéro est passé dans *aprèsLigne*.

Si vous ne passez pas le paramètre *process*, **INSERER LIGNE MENU** est appliquée à la barre de menus du process courant. Sinon, **INSERER LIGNE MENU** est appliquée à la barre de menus du process dont la référence est passée dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Si vous ne passez pas le paramètre *, **INSERER LIGNE MENU** vous permet d'insérer une ou plusieurs lignes de menus en une seule fois.

INSERER LIGNE MENU fonctionne comme **AJOUTER LIGNE MENU**, hormis le fait qu'elle permet d'insérer des commandes de menu partout dans le menu alors que **AJOUTER LIGNE MENU** les ajoute toujours à la fin du menu.

Reportez-vous à la description de la commande **AJOUTER LIGNE MENU** pour plus de détails sur la définition des commandes de menus passée dans *libelléLigne* et sur l'action du paramètre *.

Le paramètre facultatif *sousMenu* vous permet de désigner un menu comme ligne insérée et donc de définir un sous-menu hiérarchique. Vous devez passer dans ce paramètre une référence de menu (chaîne de type *RefMenu*) désignant un menu créé par exemple à l'aide de la commande **Creer menu**. Si la commande insère plusieurs lignes de menus, le sous-menu est associé à la première ligne.

Important : Les nouvelles lignes n'ont pas de méthodes ou d'actions associées. Vous devez leur associer une action ou une méthode via les commandes **FIXER PROPRIETE LIGNE MENU** ou **FIXER METHODE LIGNE MENU** ou encore les gérer à partir d'une méthode formulaire qui utilise la fonction **Menu choisi**.

Exemple

L'exemple suivant crée un menu constitué de deux commandes auxquelles il affecte une méthode :

```
refMenu:=Creer menu
AJOUTER LIGNE MENU(refMenu;"Caractères")
FIXER METHODE LIGNE MENU(refMenu;1;"GestCaracDial")
INSERER LIGNE MENU(refMenu;1;"Paragraphes")
FIXER METHODE LIGNE MENU(refMenu;2;"GestParDial")
```

LIRE ICONE LIGNE MENU

LIRE ICONE LIGNE MENU (menu ; ligneMenu ; reflcône {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Référence de menu ou Numéro de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
reflcône	Variable texte, Variable entier long	⇐	Nom ou numéro de l'image associée à la ligne de menu
process	Entier long	⇒	Numéro de process

Description

La commande **LIRE ICONE LIGNE MENU** retourne dans la variable *reflcône* la référence de l'icône éventuellement associée à la ligne de menu désignée par les paramètres *menu* et *ligneMenu*. Cette référence est le nom ou le numéro de l'image.

Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans *menu* un identifiant unique de menu (*RefMenu*) ou un numéro de menu. Si vous passez un identifiant unique, le paramètre *process* est inutile et sera ignoré s'il est passé. Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif *process*.

Si l'icône avait été définie à l'aide d'une image de la bibliothèque, la commande retourne dans *reflcône* le nom ou le numéro de l'image en fonction du type de la variable passée dans ce paramètre. Si l'icône avait été définie à l'aide d'une image stockée dans le dossier **Resources** de la base, la commande retourne dans *reflcône* le chemin d'accès de l'image.
Si vous n'attribuez pas de type spécifique à la variable *reflcône*, par défaut le nom de l'image est retourné (type texte).

Si aucune icône n'est associée à la ligne, la commande retourne une valeur vide.

LIRE LIGNES MENU

LIRE LIGNES MENU (menu ; tabTitresMenus ; tabRefsMenus)

Paramètre	Type		Description
menu	Entier long, RefMenu	⇨	Référence de menu ou Numéro de menu
tabTitresMenus	Tableau chaîne	⇦	Tableau des libellés du menu
tabRefsMenus	Tableau chaîne	⇦	Tableau des références du menu

Description

La commande **LIRE LIGNES MENU** retourne dans les tableaux *tabTitresMenu* et *tabRefsMenu* les libellés et les identifiants de toutes les lignes du menu ou de la barre de menus désigné(e) par le paramètre *menu*.

Vous pouvez passer dans *menu* un identifiant unique de menu (*RefMenu*), un numéro de barre de menus ou une référence de barre de menus obtenue via la commande **Lire reference barre menu**.

Lorsqu'aucune référence de menu n'est rattachée à une ligne, une chaîne vide est retournée dans l'élément de tableau correspondant.

Exemple

Vous souhaitez connaître le contenu de la barre de menus du process courant :

```
TABLEAU TEXTE(tabTitresMenu;0)
TABLEAU TEXTE(tabRefsMenu;0)
RefBarreMenu:=Lire reference barre menu(Process de premier plan)
LIRE LIGNES MENU(RefBarreMenu;tabTitresMenu;tabRefsMenu)
```

Lire marque ligne menu

Lire marque ligne menu (menu ; ligneMenu {; process}) -> Résultat

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Entier long	⇒	Numéro de référence de process
Résultat	Chaîne	⇐	Marque de ligne de menu courante

Description

La commande **Lire marque ligne menu** retourne la marque (ou "coche") de la ligne de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans *menu* et *ligneMenu*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si vous omettez le paramètre *process*, **Lire marque ligne menu** s'applique à la barre de menus du process courant. Sinon, **Lire marque ligne menu** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Si la ligne de menu n'a pas de marque ou si le paramètre *ligneMenu* désigne un sous-menu hiérarchique, **Lire marque ligne menu** retourne une chaîne vide.

Note : Pour plus d'informations sur les marques des lignes de menus sous Mac OS et Windows, reportez-vous à la description de la commande **FIXER MARQUE LIGNE MENU**.

Exemple

L'exemple suivant inverse l'état marqué d'une ligne de menu :

```
FIXER MARQUE LIGNE MENU($vlMenu;$vlItem;Caractere(18)*Num(Code de caractere(Lire marque ligne
menu($vlMenu;$vlItem))#18))
```

Lire methode ligne menu

Lire methode ligne menu (menu ; ligneMenu {; process}) -> Résultat

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Référence de menu ou Numéro de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
process	Entier long	⇒	Numéro de process
Résultat	Chaîne	⇐	Nom de la méthode

Description

La commande **Lire methode ligne menu** retourne le nom de la méthode projet 4D associée à la ligne de menu désignée par les paramètres *menu* et *ligneMenu*.
Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans *menu* un identifiant unique de menu (*RefMenu*) ou un numéro de menu. Si vous passez un identifiant unique, le paramètre *process* est inutile et sera ignoré s'il est passé. Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif *process*.

La commande retourne le nom de la méthode 4D sous la forme d'une chaîne de caractères (expression). Si aucune méthode n'est associée à la ligne de menu, la commande retourne une chaîne vide.

Lire modificateurs ligne menu

Lire modificateurs ligne menu (menu ; ligneMenu {; process}) -> Résultat

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Référence de menu ouNuméro de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
process	Entier long	⇒	Numéro de process
Résultat	Entier long	⇐	Touche(s) de modification associée(s) à la ligne de menu

Description

La commande **Lire modificateurs ligne menu** retourne le ou les modificateur(s) additionnel(s) associé(s) au raccourci standard de la ligne de menu désignée par les paramètres *menu* et *ligneMenu*.
Le raccourci standard est composé de la touche **Commande** (Mac OS) ou **Ctrl** (Windows) et d'une touche personnalisée. Le raccourci standard est géré via les commandes **FIXER RACCOURCI LIGNE MENU** et **Lire touche ligne menu**.

Les modificateurs additionnels sont la touche **Majuscule** et la touche **Option** (Mac OS) / **Alt** (Windows). Ces modificateurs ne sont utilisables que si un raccourci standard a été défini au préalable.

La valeur numérique retournée par la commande correspond au code de la ou des touche(s) de modification additionnelles. Les codes des touches sont les suivants :

- **Majuscule** = 512
- **Option** (Mac OS) ou **Alt** (Windows) = 2048
  Si les deux touches sont utilisées, leur valeur est cumulée.

Note : Vous pouvez évaluer la valeur retournée à l'aide des constantes Masque touche majuscule et Masque touche option du thème "**Evénements (Modifiers)**".

Si la ligne de menu n'a pas de touche de modification associée, la commande retourne 0.
Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans *menu* un identifiant unique de menu (*RefMenu*) ou un numéro de menu.
Si vous passez un identifiant unique, le paramètre *process* est inutile et sera ignoré s'il est passé.
Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif *process*.

Exemple

Reportez-vous à l'exemple de la commande **Lire touche ligne menu**.

Lire parametre ligne menu

Lire parametre ligne menu (menu ; ligneMenu) -> Résultat

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Référence de menu ou Numéro de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
Résultat	Chaîne	⇐	Paramètre personnalisé de la ligne de menu

Description

La commande **Lire parametre ligne menu** retourne la chaîne de caractères personnalisée associée à la ligne de menu désignée par les paramètres *menu* et *ligneMenu*. Cette chaîne doit avoir été préalablement définie à l'aide de la commande **FIXER PARAMETRE LIGNE MENU**.

Lire parametre ligne menu selectionnee

Lire parametre ligne menu selectionnee -> Résultat

Paramètre	Type		Description
Résultat	Chaîne		Paramètre personnalisé de la ligne de menu

Description

La commande **Lire parametre ligne menu selectionnee** retourne la chaîne de caractères personnalisée associée à la ligne de menu sélectionnée. Ce paramètre doit avoir été préalablement défini à l'aide de la commande **FIXER PARAMETRE LIGNE MENU**.
Si aucune ligne de menu n'a été sélectionnée, la commande retourne une chaîne vide "".

LIRE PROPRIETE LIGNE MENU

LIRE PROPRIETE LIGNE MENU (menu ; ligneMenu ; propriété ; valeur {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Référence de menu ou Numéro de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
propriété	Chaîne	⇒	Type de propriété
valeur	Expression	⇐	Valeur de la propriété
process	Entier long	⇒	Numéro de process

Description

La commande **LIRE PROPRIETE LIGNE MENU** retourne dans le paramètre *valeur* la valeur courante de la propriété de la ligne de menu désignée par les paramètres *menu* et *ligneMenu*.
Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans *menu* un identifiant unique de menu (*RefMenu*) ou un numéro de menu. Si vous passez un identifiant unique, le paramètre *process* est inutile et sera ignoré s'il est passé. Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif *process*.

Passez dans le paramètre *propriété* la propriété dont vous souhaitez obtenir la valeur. Vous pouvez utiliser l'une des constantes du thème "**Propriétés des lignes de menu**" ou une chaîne correspondant à une propriété personnalisée. Pour plus d'informations sur les propriétés des menus et leurs valeurs, reportez-vous à la description de la commande **FIXER PROPRIETE LIGNE MENU**.

Lire reference barre menu

Lire reference barre menu {(process)} -> Résultat

Paramètre	Type		Description
process	Entier long	⇒	Numéro de référence du process
Résultat	RefMenu	⇐	Identifiant de la barre de menus

Description

La commande **Lire reference barre menu** renvoie l'identifiant unique de la barre de menus courante ou de la barre de menus d'un process spécifique.

Si la barre de menus a été créée par la commande **Creer menu**, cet identifiant correspond à la référence unique du menu créé. Sinon, la commande retourne un identifiant interne spécifique. Dans tous les cas, cet identifiant *RefMenu* pourra être utilisé pour référencer la barre de menus par toutes les autres commandes du thème.

Le paramètre *process* permet de désigner le process duquel vous souhaitez obtenir l'identifiant de la barre de menus courante. Si vous omettez ce paramètre, la commande retourne l'identifiant de la barre de menus du process courant.

Exemple

Reportez-vous à l'exemple de la commande **LIRE LIGNES MENU**.

Lire style ligne menu

Lire style ligne menu (menu ; ligneMenu {; process}) -> Résultat

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Entier long	⇒	Numéro de référence de process
Résultat	Entier long	⇐	Style courant de la ligne de menu

Description

La commande **Lire style ligne menu** retourne le style de police de la ligne de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans *menu* et *ligneMenu*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si vous omettez le paramètre *process*, **Lire style ligne menu** s'applique à la barre de menus du process courant. Sinon, **Lire style ligne menu** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Lire style ligne menu retourne une combinaison (une ou une somme) des constantes prédéfinies suivantes, placées dans le thème **Styles de caractères** :

Constante	Type	Valeur
Gras	Entier long	1
Italique	Entier long	2
Normal	Entier long	0
Souligné	Entier long	4

Exemple

Si, par exemple, vous voulez tester si une ligne de menu est affichée en gras, vous écrivez :

```
Si((Lire style ligne menu($vlMenu;$vlItem)&Gras)#0)
  `...
Fin de si
```

Lire texte ligne menu

Lire texte ligne menu (menu ; ligneMenu {; process}) -> Résultat

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Entier long	⇒	Numéro de référence de process
Résultat	Chaîne	⇐	Libellé de la ligne de menu

Description

La commande **Lire texte ligne menu** retourne le libellé de la commande de menu dont le numéro ou la référence de menu et le numéro de commande ont été passés dans *menu* et *ligneMenu*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si vous ne passez pas le paramètre *process*, **Lire texte ligne menu** est appliquée à la barre de menus du process courant. Sinon **Lire texte ligne menu** est appliquée à la barre de menus du process dont la référence est passée dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Lire titre menu

Lire titre menu (menu {; process}) -> Résultat

Paramètre	Type		Description
menu	Entier long, RefMenu	➡	Numéro de menu ou Référence de menu
process	Entier long	➡	Numéro de référence de process
Résultat	Chaîne	⮌	Titre du menu

Description

La commande **Lire titre menu** retourne le titre du menu dont vous avez passé le numéro ou la référence dans *menu*.

Si vous omettez le paramètre *process*, **Lire titre menu** s'applique à la barre de menus du process courant. Sinon, **Lire titre menu** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Lire touche ligne menu

Lire touche ligne menu (menu ; ligneMenu {; process}) -> Résultat

Paramètre	Type		Description
menu	Entier long, RefMenu	➡	Numéro de menu ou Référence de menu
ligneMenu	Entier long	➡	Numéro de la ligne de menu ou -1 pour la dernière ligne ajoutée
process	Entier long	➡	Numéro de référence de process
Résultat	Entier long	⮌	Code de caractère de de la touche de raccourci standard associée à la ligne de menu

Description

La commande **Lire touche ligne menu** retourne le code de la touche **Ctrl** (sous Windows) ou **Commande** (Mac OS) utilisée comme raccourci clavier pour la commande de menu dont le numéro ou la référence de menu et le numéro de ligne ont été passés dans *menu* et *ligneMenu*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si vous ne passez pas le paramètre *process*, **Lire touche ligne menu** est appliquée à la barre de menus du process courant. Sinon, **Lire touche ligne menu** est appliquée à la barre de menus du process dont la référence est passée dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Si la ligne de menu n'a pas de touche de raccourci associée ou si le paramètre *ligneMenu* désigne un sous-menu hiérarchique, **Lire touche ligne menu** retourne *0* (zéro).

Exemple

Pour obtenir le raccourci clavier associé à une ligne de menu, il est utile de mettre en place une structure de programmation du type suivant :

```
Si(Lire touche ligne menu(monmenu;1)#0)
   $modifiers:=Lire modificateurs ligne menu(monmenu;1)
   Au cas ou
      :($modifiers=Masque touche option)
         ...
      :($modifiers=Masque touche majuscule)
         ...
      :($modifiers=Masque touche option+Masque touche majuscule)
         ...
   Fin de cas
Fin de si
```

Menu choisi

Menu choisi {(sousMenu)} -> Résultat

Paramètre	Type		Description
sousMenu	RefMenu	⇐	Référence du menu contenant la ligne sélectionnée
Résultat	Entier long	⇒	Commande de menu sélectionnée : Mot machine haut = n° de menu, Mot machine bas = n° de commande de menu

Description

Menu choisi ne s'utilise que lorsqu'un formulaire est affiché. Cette fonction détecte la commande de menu choisie dans un menu et, dans le cas d'un sous-menu hiérarchique, retourne la référence du sous-menu.

Astuce : A chaque fois que cela est possible, utilisez des méthodes associées à des commandes de menus dans une barre associée (avec un numéro de barre négatif) plutôt que d'appeler **Menu choisi**. Les barres de menus associées sont plus faciles à gérer, puisqu'il n'est pas nécessaire de tester leur sélection.

La commande **Menu choisi** permet de travailler avec des sous-menus hiérarchiques. En cas de sélection d'une ligne d'un menu hiérarchique au-delà du premier niveau, la commande retourne dans le paramètre facultatif *sousMenu* la référence (type RefMenu, chaîne de 16 caractères) du sous-menu auquel appartient la ligne sélectionnée. Si la commande du menu ne contient pas de sous-menu hiérarchique, ce paramètre reçoit une chaîne vide.

Menu choisi retourne le numéro système du menu sélectionné sous forme d'Entier long. Pour obtenir le numéro du menu, divisez **Menu choisi** par 65536 et convertissez le résultat en Entier. Pour obtenir le numéro de la commande de menu, calculez le modulo de **Menu choisi** avec le coefficient 65536. Utilisez les formules suivantes pour calculer le numéro du menu et de la commande de menu :

```
Menu:=Menu choisi\ 65536
Ligne de menu:=Menu choisi% 65536
```

Vous pouvez également extraire ces valeurs à l'aide des **Opérateurs sur les bits**, comme dans l'exemple suivant :

```
Menu:=(Menu choisi & 0xFFFF0000)>>16
Ligne de menu:=Menu choisi & 0xFFFF
```

Menu choisi retourne 0 si aucune commande de menu n'est sélectionnée.

Exemple

La méthode formulaire suivante utilise la fonction **Menu choisi** pour fournir les arguments "menu" et "ligne de menu" à **FIXER MARQUE LIGNE MENU** :

```
Au cas ou
   :(Evenement formulaire=Sur menu sélectionné)
      C_ALPHA(16;$refMenuIncludingItem)
      C_ENTIER LONG($ref;$NumMenu;$NumMenuItem)
      $ref:=Menu choisi($refMenuIncludingItem)
      $NumMenu:=$ref\65536
      $NumMenuItem:=$ref%65536
      FIXER MARQUE LIGNE MENU($refMenuIncludingItem;$NumMenuItem;Caractere(18))
Fin de cas
```

Note : L'événement Sur menu sélectionné n'est pas activé si aucune ligne n'est sélectionnée, *$refmenuincludingItem* est toujours renseigné et *$NumMenu* vaut 0 si le menu n'est pas un des menus de la barre.

Nombre de lignes de menu

Nombre de lignes de menu (menu {; process}) -> Résultat

Paramètre	Type		Description
menu	Entier long, RefMenu	➡	Numéro de menu ou Référence de menu
process	Entier long	➡	Numéro de référence de process
Résultat	Entier long	⮌	Nombre de lignes du menu

Description

La commande **Nombre de lignes de menu** retourne le nombre de lignes (commandes) de menus présentes dans le menu dont vous avez passé le numéro ou la référence dans *menu.*

Si vous omettez le paramètre *process*, **Nombre de lignes de menu** s'applique à la barre de menus du process courant. Sinon, **Nombre de lignes de menu** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Nombre de menus

Nombre de menus {(process)} -> Résultat

Paramètre	Type		Description
process	Entier long	→	Numéro de référence de process
Résultat	Entier long	↩	Nombre de menus de la barre de menus courante

Description

La commande **Nombre de menus** retourne le nombre de menus présents dans la barre de menus.

Si vous omettez le paramètre *process*, **Nombre de menus** s'applique à la barre de menus du process courant. Sinon, **Nombre de menus** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

SUPPRIMER LIGNE MENU

SUPPRIMER LIGNE MENU (menu ; ligneMenu {; process})

Paramètre	Type		Description
menu	Entier long, RefMenu	⇒	Numéro de menu ou Référence de menu
ligneMenu	Entier long	⇒	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Entier long	⇒	Numéro de référence de process

Description

La commande **SUPPRIMER LIGNE MENU** supprime la ligne de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans *menu* et *ligneMenu*. Vous pouvez passer -1 dans *ligneMenu* afin de désigner la dernière ligne ajoutée au *menu*.

Si la ligne de menu désignée par *menu* et *ligneMenu* est elle-même un menu géré par référence et créé par exemple à l'aide la commande **Creer menu**, **SUPPRIMER LIGNE MENU** supprimera uniquement l'instance de *ligneMenu* dans *menu*. Le sous-menu référencé par *ligneMenu* continuera d'exister en mémoire. Vous devez utiliser la commande **EFFACER MENU** afin de supprimer définitevement un menu géré par référence.
Cette commande fonctionne également avec une barre de menus créée avec la commande **Creer menu** et installée avec la commande **FIXER BARRE MENUS**.

Si vous omettez le paramètre *process*, **SUPPRIMER LIGNE MENU** s'applique à la barre de menus du process courant. Sinon, **SUPPRIMER LIGNE MENU** s'applique à la barre de menus du process dont vous avez passé le numéro dans *process*.

Note : Si vous passez un paramètre *RefMenu* dans *menu*, le paramètre *process* est inutile et sera ignoré.

Note : Pour soigner l'ergonomie de votre interface, ne laissez pas accessible un menu ne comportant aucune ligne.

Messages

- AFFICHER NOTIFICATION
- ALERTE
- CONFIRMER
- Demander
- LAISSER MESSAGES
- MESSAGE
- POSITION MESSAGE
- SUPPRIMER MESSAGES

AFFICHER NOTIFICATION

AFFICHER NOTIFICATION (titre ; contenu {; durée})

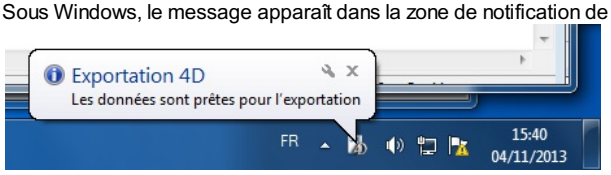
Paramètre	Type		Description
titre	Alpha	➡	Titre de la notification
contenu	Alpha	➡	Texte de la notification
durée	Entier long	➡	Délai d'affichage en secondes

Description

La commande **AFFICHER NOTIFICATION** provoque l'affichage d'un message de notification à destination de l'utilisateur.

Ce type de message est généralement utilisé par le système ou les applications pour informer l'utilisateur d'un événement (déconnexion réseau, disponibilité de mises à jour, etc.)

- Sous Windows, le message apparaît dans la zone de notification de la barre des tâches :



- Sous OS X (version 10.8 minimum), le message apparaît dans une petite fenêtre glissant dans l'angle supérieur droit de l'écran.

A noter que, conformément aux spécifications d'Apple, la notification n'est affichée que si l'application n'est pas au premier plan. Le message apparaît cependant toujours dans la liste du "notification center".

Passez dans les paramètres *titre* et *contenu* le titre et le texte du message à afficher (dans notre exemple, le titre est "Exportation 4D"). Vous pouvez saisir jusqu'à 255 caractères.

Sous Windows, la fenêtre du message reste affichée tant qu'aucune activité n'a été détectée sur la machine, ou jusqu'à ce que l'utilisateur clique sur sa case de fermeture. Le paramètre facultatif *durée* permet de modifier la durée d'affichage par défaut. A noter que l'affichage des notifications dépend des configurations système.

Exemple

```
AFFICHER NOTIFICATION("Exportation 4D";"Les données sont prêtes pour l'exportation")
```

ALERTE

ALERTE (message ; libelléBoutonOK ; test)

Paramètre	Type		Description
message	Chaîne	➡	Message à afficher dans la boîte de dialogue d'alerte
libelléBoutonOK	Chaîne	➡	Libellé du bouton OK
test	Tableau texte 2D	➡	entrée plat dessert

Description

La commande **ALERTE** affiche une boîte de dialogue d'alerte composée d'une icône, d'un message et d'un bouton OK.

Vous passez le message à afficher dans le paramètre *message*. Ce message peut contenir jusqu'à 255 caractères. Si la longueur ou la largeur des caractères est trop importante par rapport à la zone du message, il sera tronqué.

Par défaut, le libellé du bouton OK est “OK”. Si vous voulez changer ce libellé, passez le nouveau libellé dans le paramètre optionnel *libelléBoutonOK*. Si nécessaire, la largeur du bouton OK est augmentée vers la gauche pour contenir ce nouveau libellé.

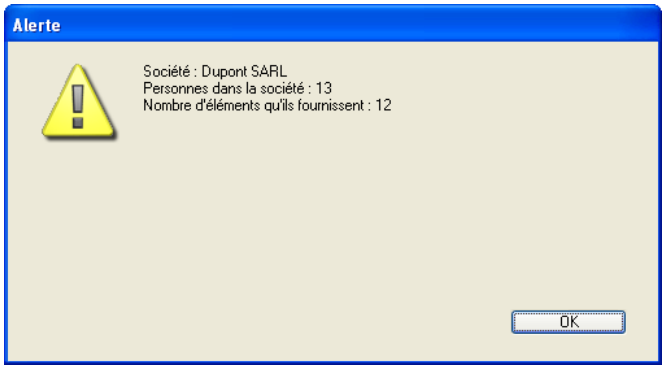
Note : N'appelez pas la commande **ALERTE** dans une méthode formulaire ou une méthode objet qui gère l'événement formulaire Sur activation ou Sur désactivation, car cela provoquerait une boucle sans fin.

Exemple 1

L'exemple suivant appelle une boîte de dialogue d'alerte qui affiche des informations sur une société. Notez que le message contient des retours chariot (Caractere(13)) qui forcent le texte à passer sur la ligne suivante :

```
ALERTE("Société : "+[Sociétés]Nom+Caractere(13)+"Personnes dans la société : "+
Chaine(Enregistrements trouves([Personne]))+Caractere(13)+"Nombre
d'éléments qu'ils fournissent"+Chaine (Enregistrements trouves([Eléments])))
```

Voici la boîte de dialogue d'alerte affichée (sous Windows) par notre exemple :

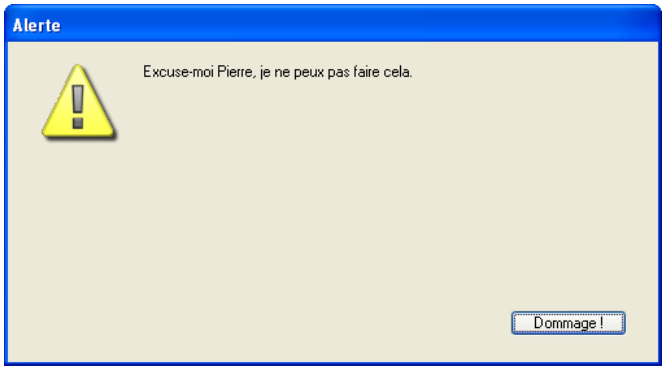


Exemple 2

Voici un autre exemple :

```
ALERTE("Excuse-moi Pierre, je ne peux pas faire cela.";"Dommage !")
```

Cette instruction affichera (sous Windows) la boîte de dialogue d'alerte suivante :

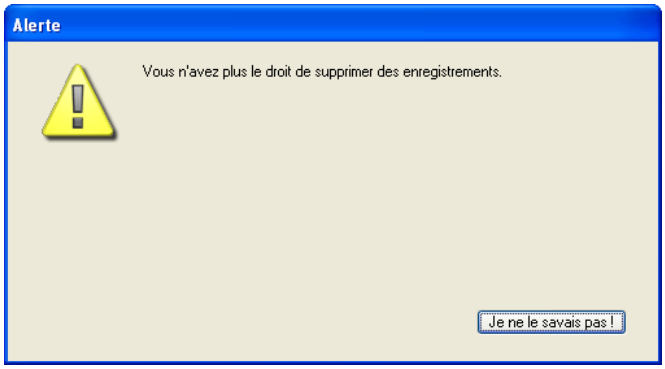


Exemple 3

Voici un autre exemple :

```
ALERTE("Vous n'avez plus le droit de supprimer des enregistrements.";"Je ne le savais pas !")
```

Ce code affiche la boîte de dialogue d'alerte suivante :


Alerte
Vous n'avez plus le droit de supprimer des enregistrements.
Je ne le savais pas !

CONFIRMER

CONFIRMER (message {; libelléBoutonOK {; libelléBoutonAnn}})

Paramètre	Type		Description
message	Chaîne	⇒	Message à afficher dans la boîte de dialogue de confirmation
libelléBoutonOK	Chaîne	⇒	Libellé du bouton OK
libelléBoutonAnn	Chaîne	⇒	Libellé du bouton Annuler

Description

La commande **CONFIRMER** affiche une boîte de dialogue de confirmation qui se compose d'une icône, d'un message, d'un bouton OK et d'un bouton Annuler.

Les boîtes de dialogue de confirmation ou d'alerte sont utilisées pour afficher des informations (comme des messages d'erreur) qui ne nécessitent pas d'informations en retour.

Vous passez le message à afficher dans le paramètre *message*. Ce message peut contenir jusqu'à 255 caractères. Si la longueur ou largeur des caractères est trop importante par rapport à la zone d'affichage, le message sera tronqué.

Par défaut, le libellé du bouton OK est “OK” et le libellé du bouton Annuler est “Annuler”. Si vous voulez modifier le libellé de ces boutons, passez le nouveau libellé dans les paramètres optionnels *libelléBoutonOK* et *libelléBouton*. Si nécessaire, les boutons sont agrandis vers la gauche en fonction de la taille des libellés que vous avez saisis.

Le bouton OK est le bouton par défaut. L'utilisateur peut cliquer sur le bouton OK ou appuyer sur la touche **Entrée** pour valider la boîte de dialogue, la variable système OK prend alors la valeur 1. L'utilisateur peut cliquer sur le bouton Annuler pour annuler la boîte de dialogue, la variable système OK prend alors la valeur 0.

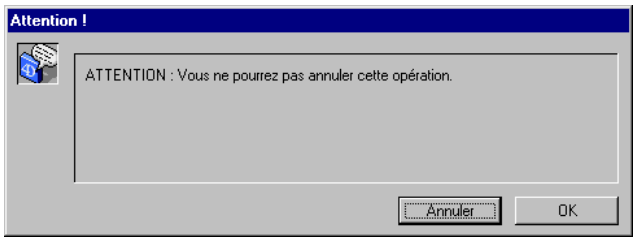
Conseil : N'appelez pas la commande **CONFIRMER** dans une méthode formulaire ou objet qui gère l'événement Sur activation ou Sur désactivation, car cela provoquerait une boucle sans fin.

Exemple 1

L'exemple ci-dessous :

```
CONFIRMER("ATTENTION : Vous ne pourrez pas annuler cette opération.")
Si(OK=1)
   TOUT SELECTIONNER([Vieilles choses])
   SUPPRIMER SELECTION([Vieilles choses])
Sinon
   ALERTE("Opération annulée.")
Fin de si
```

... provoquera l'affichage de la boîte de dialogue de confirmation suivante (sous Windows) :

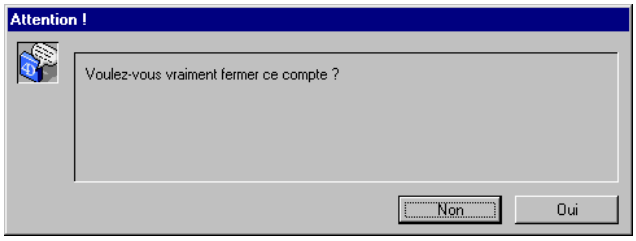


Exemple 2

La ligne :

```
CONFIRMER("Voulez-vous vraiment fermer ce compte ?";"Oui";"Non")
```

... provoquera l'affichage de la boîte de dialogue de confirmation suivante (sous Windows) :

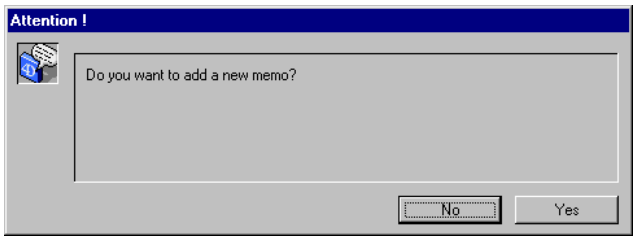


Exemple 3

Vous développez une application 4D pour le marché international. Vous avez écrit une méthode projet qui retourne du texte traduit à partir d'une version française. Vous avez également rempli un tableau nommé *asLocalizedUIMessages* dans lequel vous stockez les mots les plus courants. Dans ce cas, la ligne :

```
CONFIRMER(INTL Text("Voulez-vous ajouter un nouveau mémo
?");asLocalizedUIMessages{kLoc_OUI};asLocalizedUIMessages{kLoc_NON})
```

... pourrait afficher la boîte de dialogue de confirmation (sous Windows) suivante :



Exemple 4

La ligne :

```
CONFIRMER("ATTENTION : Si vous poursuivez cette opération, des enregistrements seront "+"modifiés
irrémédiablement."+Caractere(13)+"Que voulez-vous faire ?";"Ne PAS continuer";"Continuer")
```

... provoque l'affichage de la boîte de dialogue de confirmation suivante (sous Mac OS) :

Demander

Demander (message {; réponseDéfaut {; titreBoutonOK {; titreBoutonAnn}}}) -> Résultat

Paramètre	Type		Description
message	Chaîne	➡	Message à afficher dans la boîte de dialogue
réponseDéfaut	Chaîne	➡	Valeur par défaut dans la zone de saisie de texte
titreBoutonOK	Chaîne	➡	Libellé du bouton OK
titreBoutonAnn	Chaîne	➡	Libellé du bouton Annuler
Résultat	Chaîne	⬅	Valeur saisie par l'utilisateur

Description

La fonction **Demander** affiche une boîte de dialogue de demande d'informations composée d'un message, d'une zone de saisie de texte, d'un bouton **OK** et d'un bouton **Annuler**.

Vous passez le message à afficher dans le paramètre *message*. Si la taille du message excède les capacités de la zone d'affichage (aux alentours de 50 caractères, variant en fonction du Système et de la police utilisée), il peut apparaître tronqué.

Par défaut, le libellé du bouton OK est “OK” et celui du bouton Annuler est “Annuler”. Si vous voulez modifier ces libellés, passez d'autres valeurs dans les paramètres optionnels *titreBoutonOK* et *titreBoutonAnn*. Si nécessaire, les boutons sont agrandis vers la gauche, en fonction de la taille des libellés que vous avez saisis.

Le bouton OK est le bouton par défaut. L'utilisateur peut cliquer sur le bouton OK ou appuyer sur la touche Entrée pour valider la boîte de dialogue, mettant ainsi la variable système OK à 1. Il peut également cliquer sur le bouton Annuler pour annuler la boîte de dialogue, mettant ainsi la variable système OK à 0.

L'utilisateur peut taper des caractères dans la zone de saisie de texte. Pour définir une valeur par défaut, passez le texte par défaut dans le paramètre *réponseDéfaut*. Si l'utilisateur clique sur le bouton OK, **Demander** retourne le texte. Si l'utilisateur clique sur le bouton Annuler, **Demander** retourne une chaîne vide (""). Si la réponse doit être une valeur numérique ou une date, convertissez la chaîne retournée par **Demander** dans le type souhaité à l'aide des fonctions **Num** et **Date**.

Note : N'appelez pas la fonction **Demander** dans une méthode formulaire ou objet qui gère l'événement Sur activation ou Sur désactivation car cela provoquerait une boucle sans fin.

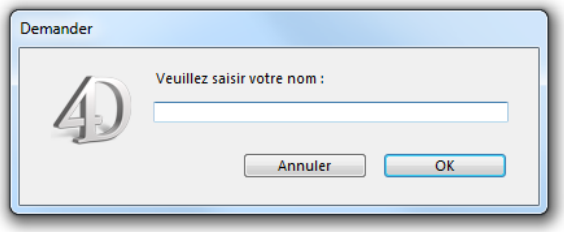
Conseil : Si vous voulez récupérer plusieurs informations de l'utilisateur, construisez un formulaire approprié et appelez-le avec la commande **DIALOGUE**, plutôt que d'afficher une succession de boîtes de dialogue du type **Demander**.

Exemple 1

La ligne de code :

```
$vsAffiche :=Demander("Veuillez saisir votre nom :")
```

... provoquera l'affichage de la boîte de dialogue suivante :

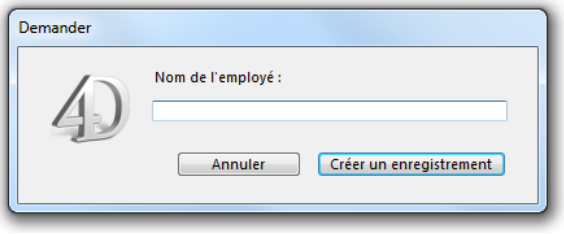


Exemple 2

Le code suivant :

```
$vsAffiche:=Demander("Nom de l'employé :";"";"Créer un enregistrement";"Annuler")
Si(OK=1)
   AJOUTER ENREGISTREMENT([Employés])
   ` Note : $vsAffiche est alors copiée dans le champ [Employés]Nom
   ` lors de l'événement formulaire Sur chargement de la méthode formulaire
Fin de si
```

... provoquera l'affichage de la boîte de dialogue suivante :

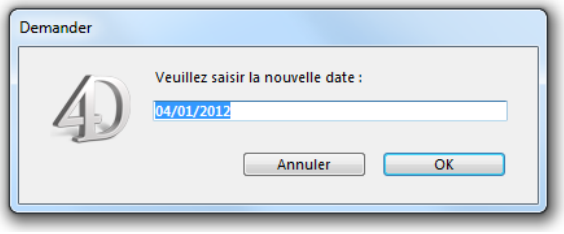


Exemple 3

La ligne de code :

```
$vdAffiche:=Date(Demander("Veuillez saisir la nouvelle date :";Chaine(Date du jour)))
```

... provoquera l'affichage de la boîte de dialogue suivante :



LAISSER MESSAGES

LAISSER MESSAGES
Ne requiert pas de paramètre

Description

Reportez-vous à la description de la commande **SUPPRIMER MESSAGES**.

MESSAGE

MESSAGE (message)

Paramètre	Type		Description
message	Chaîne	⇨	Message à afficher

Description

La commande **MESSAGE** affiche *message* à l'écran dans une fenêtre spéciale de message qui est ouverte et refermée à chaque fois que vous l'appelez (à moins que vous ne travailliez dans une fenêtre préalablement ouverte par la commande **Creer fenetre**, cf. ci-dessous). Le message est temporaire et est effacé dès qu'un formulaire est affiché ou dès que l'exécution de la méthode est stoppée. Si une autre commande **MESSAGE** est exécutée, le précédent message est effacé.

MESSAGE est généralement utilisée pour informer l'utilisateur du déroulement d'une action.

Si une fenêtre a été ouverte par la commande **Creer fenetre**, tous les appels ultérieurs à la commande **MESSAGE** affichent les messages dans cette fenêtre. Cette fenêtre se comporte en quelque sorte comme un terminal :

- Chaque message successif n'efface pas le précédent, les messages se placent les uns à la suite des autres.
- Si un message est plus large que la fenêtre, 4D insère automatiquement un retour à la ligne.
- Si le message contient plus de lignes que ne peut en afficher la fenêtre, 4D fait automatiquement défiler le message dans la fenêtre.
- Si vous souhaitez contrôler les retours à la ligne, insérez vos propres retours chariot dans votre texte, à l'aide de **Caractere(13)**.
- Vous pouvez appeler la commande **POSITION MESSAGE** pour afficher le texte à un emplacement particulier dans la fenêtre.
- Vous pouvez appeler la commande **EFFACER FENETRE** pour effacer le contenu de la fenêtre.
- La fenêtre est une fenêtre d'affichage statique : son contenu n'est pas redessiné lorsque d'autres fenêtres s'affichent par-dessus.
- La police et la taille des caractères affichés dans la fenêtre peuvent être modifiées via la page "Interface" des Propriétés de la base.

Note : MESSAGE est compatible avec la commande **Creer fenetre formulaire**, toutefois dans ce contexte le second paramètre * de **Creer fenetre formulaire**, permettant de conserver la taille et position de la fenêtre, n'est pas pris en charge.

Exemple 1

L'exemple suivant traite une sélection d'enregistrements et appelle la commande **MESSAGE** pour informer l'utilisateur de la progression de l'opération :

```
Boucle($vlEnregistrement;1;Enregistrements trouves([touteTable]))
   MESSAGE("Traitement de l'enregistrement "+Chaine($vlEnregistrement))
   ` Faire quelque chose avec l'enregistrement
   ENREGISTREMENT SUIVANT([touteTable])
Fin de boucle
```

La fenêtre suivante s'affiche puis disparaît à chaque appel de **MESSAGE** :

Exemple 2

Afin d'éliminer le "clignotement" de la fenêtre, il est préférable, comme dans ce deuxième exemple, d'afficher les messages dans une fenêtre ouverte par l'intermédiaire de la commande **Creer fenetre** :

```
Creer fenetre(50;50;500;250;5;"Opération en cours")
Boucle($vlEnregistrement;1;Enregistrements trouves([touteTable]))
   MESSAGE("Traitement de l'enregistrement "+Chaine($vlEnregistrement))
   ` Faire quelque chose avec l'enregistrement
   ENREGISTREMENT SUIVANT([touteTable])
Fin de boucle
FERMER FENETRE
```

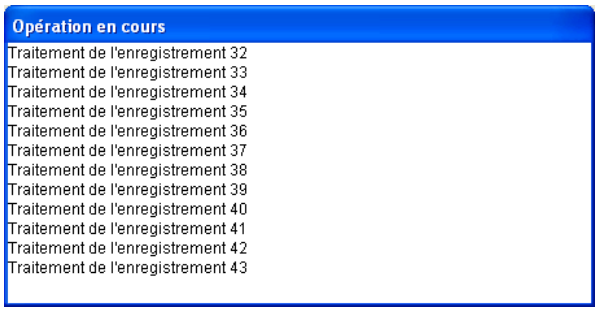
Le résultat est le suivant (sous Windows) :

Exemple 3

En ajoutant un retour chariot, vous améliorez la présentation :

```
Creer fenetre(50;50;500;250;5;"Opération en cours")
Boucle($vlEnregistrement;1;Enregistrements trouves([touteTable]))
   MESSAGE("Traitement de l'enregistrement "+Chaine($vlEnregistrement)+Caractere(Retour chariot))
   ` Faire quelque chose avec l'enregistrement
   ENREGISTREMENT SUIVANT([touteTable])
Fin de boucle
FERMER FENETRE
```

Voici le résultat (Sous Windows) :

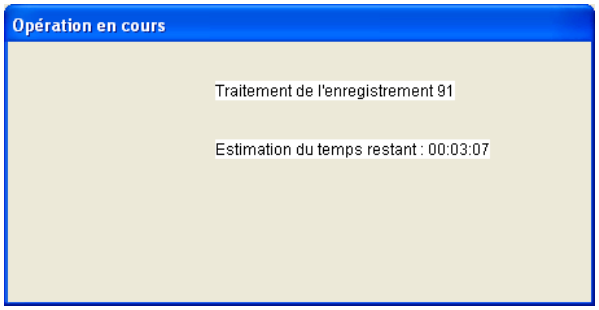


Exemple 4

A l'aide de la commande **POSITION MESSAGE** et de l'écriture de quelques lignes supplémentaires, la présentation s'améliore nettement :

```
Creer fenetre(50;50;500;250;5;"Opération en cours")
$vlNbEnregistrements:=Enregistrements trouves([touteTable])
$vhHeureDébut:=Heure courante
Boucle($vlEnregistrement;1;$vlNbEnregistrements)
   POSITION MESSAGE(5;2)
   MESSAGE("Traitement de l'enregistrement "+Chaine($vlEnregistrement)+Caractere(Retour chariot))
   ` Faire quelque chose avec les enregistrements
   ENREGISTREMENT SUIVANT([touteTable])
   POSITION MESSAGE(5;5)
   $vlReste:=(($vlNbEnregistrements/$vlEnregistrement)-1)*(Heure courante-$vhHeureDébut)
   MESSAGE("Estimation du temps restant : "+Chaine heure($vlReste))
Fin de boucle
FERMER FENETRE
```

Voici le résultat (sous Windows) :



POSITION MESSAGE

POSITION MESSAGE (x ; y)

Paramètre	Type		Description
x	Entier long	⇒	Coordonnée x (horizontale) du curseur
y	Entier long	⇒	Coordonnée y (verticale) du curseur

Description

La commande **POSITION MESSAGE** est destinée à être utilisée conjointement avec la commande **MESSAGE** lorsque vous affichez des messages dans une fenêtre ouverte par la commande **Creer fenetre**.

La commande **POSITION MESSAGE** détermine l'emplacement du curseur d'insertion des caractères (ce curseur est invivible) : elle définit les coordonnées auxquelles le prochain message s'affichera à l'intérieur de la fenêtre.

L'angle supérieur gauche de la fenêtre représente les coordonnées 0,0. Le curseur est automatiquement positionné à 0,0 lorsqu'une fenêtre est créée ou après l'exécution de la commande **EFFACER FENETRE**.

Après que **POSITION MESSAGE** ait défini l'emplacement du curseur, la commande **MESSAGE** peut être appelée pour afficher des caractères dans la fenêtre.

Exemple 1

Reportez-vous à l'exemple de la commande **MESSAGE**.

Exemple 2

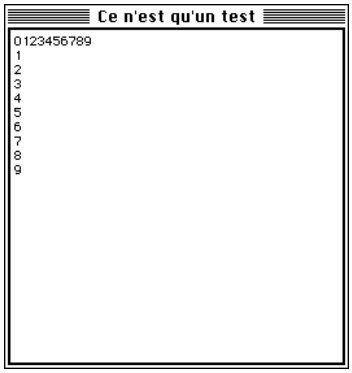
Reportez-vous à l'exemple de la fonction **Nombre de millisecondes**.

Exemple 3

L'exemple ci-dessous :

```
Creer fenetre(50;50;300;300;5;"Ce n'est qu'un test")
Boucle($vlColonne;0;9)
   POSITION MESSAGE($vlColonne;0)
   MESSAGE(Chaine($vlColonne))
Fin de boucle
Boucle($vlLigne;0;9)
   POSITION MESSAGE(0;$vlLigne)
   MESSAGE(Chaine($vlLigne))
Fin de boucle
$vhHeureDébut:=Heure courante
Repeter
Jusque((Heure courante-$vhHeureDébut)>?00:00:30?)
```

... affiche la fenêtre suivante (sous Mac OS) pendant 30 secondes :



SUPPRIMER MESSAGES

SUPPRIMER MESSAGES
Ne requiert pas de paramètre

Description

Les commandes **SUPPRIMER MESSAGES** et **LAISSER MESSAGES** suppriment ou font apparaître les thermomètres de progression affichés par 4D lorsque le programme exécute des opérations de longue durée. Par défaut, les messages sont affichés.

Voici la liste des opérations qui peuvent provoquer l'affichage d'un thermomètre de progression : Application d'une formule, Génération d'un état rapide, Export de données, Import de données, Tri, Génération d'un graphe, Recherche, Recherche par formulaire, Recherche par formule.

Voici les commandes qui peuvent provoquer l'affichage d'un thermomètre de progression :

APPLIQUER A SELECTION
CHERCHER
CHERCHER DANS SELECTION
CHERCHER PAR EXEMPLE
CHERCHER PAR FORMULE
CHERCHER PAR FORMULE DANS SELECTION
ECRITURE DIF
ECRITURE SYLK
EXPORTER TEXTE
GENERER APPLICATION
_o_GRAPHE SUR SELECTION
IMPORTER TEXTE
JOINTURE
LECTURE DIF
LECTURE SYLK
Max
Min
Moyenne
QR ETAT
REDUIRE SELECTION
SCAN INDEX
SELECTION RETOUR
Somme
TRIER
TRIER PAR FORMULE
VALEURS DISTINCTES

Note 4D Server : A compter de 4D Server v14 R3, les fenêtres de messages de progression ne sont plus affichées sur le serveur, ces opérations étant automatiquement listées dans la **Page Moniteur temps réel** de la fenêtre d'administration. Si vous souhaitez forcer l'affichage de ces fenêtres de progression, vous devez appeler la commande **LAISSER MESSAGES** sur le serveur.

Exemple

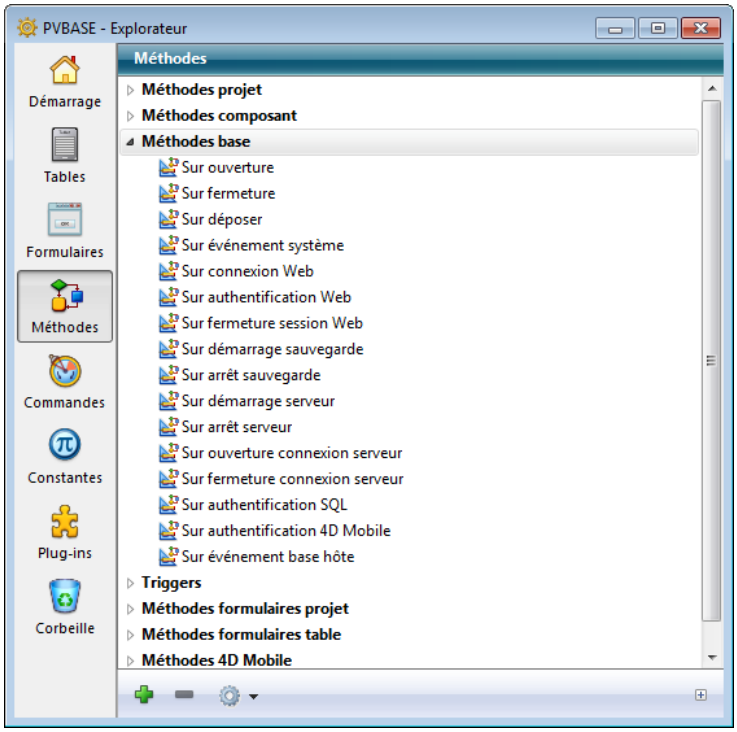
L'exemple suivant supprime les thermomètres de progression avant d'effectuer un tri, puis les rétablit après l'opération :

```
SUPPRIMER MESSAGES
TRIER([Adresses];[Adresses]CP;>;[Adresses]Nom2;>)
LAISSER MESSAGES
```

Méthodes base

Présentation des méthodes base

Les méthodes base sont des méthodes automatiquement exécutées par 4D lors d'un événement affectant la session dans sa généralité.



Pour créer, ouvrir ou éditer une méthode base :

1. Ouvrez la fenêtre de l'**Explorateur**.
2. Sélectionnez la page **Méthodes**.
3. Déployez le thème **Méthodes base**.
4. Double-cliquez sur la méthode.

ou bien :

4. Sélectionnez la méthode.
5. Appuyez sur la touche **Entrée** ou **Retour chariot**.

Vous éditez une méthode base de la même manière que n'importe quelle autre méthode.

Vous ne pouvez pas appeler une méthode base depuis une autre méthode. Les méthodes base sont automatiquement exécutées par 4D à certains moments de la session de travail. Le tableau suivant résume l'exécution des méthodes base :

Méthode base	4D local	4D Server	4D distant
Sur ouverture	Oui, une fois	Non	Oui, une fois
Sur fermeture	Oui, une fois	Non	Oui, une fois
Sur déposer	Oui, plusieurs fois	Non	Oui, plusieurs fois
Sur événement système	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur connexion Web	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur authentification Web	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur fermeture session Web	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur démarrage sauvegarde	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur arrêt sauvegarde	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur démarrage serveur	Non	Oui, une fois	Non
Sur arrêt serveur	Non	Oui, une fois	Non
Sur ouverture connexion serveur	Non	Oui, plusieurs fois	Non
Sur fermeture connexion serveur	Non	Oui, plusieurs fois	Non
Sur authentification SQL	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur authentification 4D Mobile	Oui, plusieurs fois	Oui, plusieurs fois	Non
Sur événement base hôte	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois

Méthode base Sur arrêt sauvegarde

$1 -> Méthode base Sur arrêt sauvegarde

Paramètre	Type		Description
$1	Entier long	⇦	0 = sauvegarde terminée normalement, autre valeur = erreur, interruption utilisateur ou code retourné par Sur démarrage sauvegarde

---

La **Méthode base Sur arrêt sauvegarde** est appelée à chaque fois qu'une sauvegarde de la base vient de se terminer. Les causes de l'arrêt de la sauvegarde peuvent être la fin de la copie, l'interruption par l'utilisateur ou une erreur.
Tous les environnements 4D sont concernés : 4D (tous modes), 4D Server ainsi que les applications 4D compilées et fusionnées avec 4D Volume Desktop.

La **Méthode base Sur arrêt sauvegarde** permet de vérifier que la sauvegarde s'est correctement déroulée. Elle reçoit dans le paramètre *$1* une valeur indiquant le statut de la sauvegarde à l'issue de son exécution :

- Si la sauvegarde s'est terminée normalement, *$1* vaut 0.
- Si la sauvegarde a été interrompue à la suite d'une erreur ou par l'utilisateur, *$1* est différent de 0.
  - Si la sauvegarde a été stoppée par la **Méthode base Sur démarrage sauvegarde** (*$0* # 0), *$1* retourne le code effectivement retourné dans le paramètre *$0*. Ce principe vous permet de mettre en place un système de gestion d'erreurs personnalisé.
  - Si la sauvegarde a été stoppée à la suite d'une erreur, le code de l'erreur est retourné dans *$1*.

  Dans tous les cas, vous pouvez obtenir des informations sur l'erreur à l'aide de la commande **LIRE INFORMATION SAUVEGARDE**.

Note : Vous devez impérativement déclarer le paramètre *$1* (entier long) dans la méthode base :

```
C_ENTIER LONG($1)
```

Il est important de noter qu'en cas d'erreur durant la sauvegarde (disque plein, support inaccessible...), les informations relatives à l'erreur sont uniquement affichées dans le moniteur de 4D Server ou dans le CSM, et reportées dans le journal des sauvegardes. Aucune boîte de dialogue d'alerte n'est affichée et la variable *error* n'est pas modifiée. Si vous souhaitez pouvoir notifier l'administrateur qu'une erreur s'est produite, en particulier dans le contexte d'une application en mode client/serveur, il est nécessaire d'utiliser la **Méthode base Sur arrêt sauvegarde**.

Méthode base Sur arrêt serveur

Méthode base Sur arrêt serveur
Ne requiert pas de paramètre

La **Méthode base Sur arrêt serveur** est appelée une fois sur le poste serveur lorsque la base courante est refermée sur 4D Server. La **Méthode base Sur arrêt serveur** n'est appelée dans aucun environnement 4D autre que 4D Server.

Pour refermer la base courante sur le serveur, vous pouvez sélectionner la commande de menu **Fermer la base...** sur le serveur. Vous pouvez également choisir la commande **Quitter** ou appeler la commande **QUITTER 4D** au sein d'une procédure stockée exécutée sur le serveur.

Lorsque le processus de fermeture de la base a été engagé, 4D effectue les actions suivantes :

- S'il n'y a pas de **Méthode base Sur arrêt serveur**, 4D Server tue un à un chaque process en cours d'exécution, sans distinction.
- S'il existe une **Méthode base Sur arrêt serveur**, 4D Server exécute cette méthode dans un nouveau process local. Vous pouvez donc utiliser cette méthode base pour informer les autres process, via la communication interprocess, qu'ils doivent mettre fin à leur exécution. Notez que dans tous les cas 4D Server quittera en fin de compte —la **Méthode base Sur arrêt serveur** peut effectuer toutes les opérations de nettoyage et de fermeture que vous voulez, mais elle ne peut refuser de quitter, et finira par se terminer.

La **Méthode base Sur arrêt serveur** est l'emplacement idéal pour :

- Stopper les procédures stockées lancées automatiquement à l'ouverture de la base.
- Sauvegarder (localement, sur disque) des préférences ou des paramétrages à réutiliser au début de la session suivante dans la **Méthode base Sur démarrage serveur**.
- Effectuer toute autre action que vous souhaitez déclencher automatiquement à chaque fois que vous quittez la base.

Important : Si vous utilisez la **Méthode base Sur arrêt serveur** pour refermer des procédures stockées, tenez compte du fait que le serveur quitte dès la fin de l'exécution de la **Méthode base Sur arrêt serveur** (et non des procédures stockées). Si des procédures stockées tournent encore à cet instant, elles sont purement et simplement tuées.
Par conséquent, si vous voulez être certain que les procédures stockées se terminent avant que le serveur ne les tue, il faut que la **Méthode base Sur arrêt serveur** leur signale qu'elles doivent mettre fin à leur exécution (par le test d'une variable interprocess, par exemple) mais aussi qu'elle leur laisse le temps de se refermer (boucle de n secondes ou test d'une autre variable interprocess).

Si vous voulez que du code soit exécuté automatiquement sur un poste client lorsqu'un 4D distant met un terme à sa connexion au serveur, utilisez la **Méthode base Sur fermeture**.

Méthode base Sur authentification 4D Mobile

$1, $2, $3 -> Méthode base Sur authentification 4D Mobile -> $0

Paramètre	Type		Description
$1	Texte	⇐	Nom d'utilisateur
$2	Texte	⇐	Mot de passe
$3	Booléen	⇐	Vrai = mode Digest, Faux = mode Basic
$0	Booléen	⇒	Vrai = requête acceptée, Faux = requête rejetée

Description

La **Méthode base Sur authentification 4D Mobile** vous permet de contrôler de manière personnalisée l'ouverture des sessions 4D Mobile (via REST) sur 4D. Cette méthode base est principalement destinée au filtrage des connexions lors de la mise en place d'une liaison entre un Wakanda Server et 4D.

Lorsque la demande d'ouverture de session 4D Mobile provient de Wakanda Server via la méthode ***mergeOutsideCatalog()*** (cas général), les identifiants de connexion sont fournis dans l'en-tête de la requête. La **Méthode base Sur authentification 4D Mobile** est appelée afin de vous permettre d'évaluer ces identifiants. Vous pouvez utiliser la liste des utilisateurs de la base 4D ou votre propre table d'identifiants.

Important : Lorsque la **Méthode base Sur authentification 4D Mobile** est définie (c'est-à-dire, lorsqu'elle contient du code), 4D lui délègue entièrement le contrôle des requêtes 4D Mobile : le paramétrage éventuellement effectué dans le menu "Lecture/Ecriture" de la page Web/4D Mobile des propriétés de la base est ignoré (cf. manuel *Mode Développement*).

La méthode base reçoit deux paramètres de type texte (*$1* et *$2*) et un booléen (*$3*), passés par 4D, et retourne un booléen, *$0*. Vous devez déclarer ces paramètres de la manière suivante :

```
    //Méthode base Sur authentification 4D Mobile
C_TEXTE($1;$2)
C_BOOLEEN($0;$3)
 ... // Code pour la méthode
```

$1 contient le nom d'utilisateur et *$2* le mot de passe utilisés pour la connexion.

Le mot de passe (*$2*) peut être reçu soit en clair soit sous forme hachée, en fonction du mode utilisé par la requête. Pour vous permettre d'effectuer le traitement approprié, ce mode est indiqué par le paramètre *$3* :

- si le mot de passe a été envoyé en clair (mode Basic), *$3* retourne **Faux**.
- si le mot de passe a été envoyé sous forme hachée (mode Digest), *$3* retourne **Vrai**.

Lorsque la demande de connexion 4D Mobile provient de Wakanda Server, le mot de passe est toujours envoyé sous forme hachée.

Vous devez contrôler les identifiants de la connexion 4D Mobile dans la méthode base. Généralement, vous contrôlerez le nom et le mot de passe à l'aide d'une table d'utilisateurs personnalisée. Si les identifiants sont valides, passez **Vrai** dans *$0*. La requête est alors acceptée, 4D l'exécute et retourne le résultat en JSON.
Sinon, passez **Faux** dans $0, dans ce cas la connexion est rejetée et le serveur retournera une erreur d'authentification à l'expéditeur de la requête.

Si l'utilisateur est référencé dans la liste des utilisateurs 4D de la base, vous pouvez contrôler directement le mot de passe à l'aide de l'instruction suivante :

```
$0:=Valider mot de passe($1;$2;$3)
```

La commande **Valider mot de passe** accepte un nom d'utilisateur en premier paramètre ainsi qu'un paramètre optionnel indiquant si le mot de passe est exprimé sous forme hachée.

Si vous souhaitez utiliser votre propre liste d'utilisateurs extérieurement à la liste de la base 4D, vous pouvez stocker leurs mots de passe sous une forme hachée en utilisant le même algorithme que celui utilisé par Wakanda Server lors de l'envoi de la requête de connexion à la **Méthode base Sur authentification 4D Mobile** dans *$2*. Pour hacher un mot de passe selon cette méthode, il suffit d'écrire :

```
$MdPHaché :=Generer digest($MdPClair ;Digest 4D)
```

La commande **Generer digest** accepte Digest 4D comme algorithme de hachage, correspondant à la méthode utilisée par 4D pour sa gestion interne des mots de passe.

Exemple 1

Cet exemple n'accepte que l'utilisateur "admin" avec le mot de passe "123" ne correspondant pas à un utilisateur 4D :

```
    //Méthode base sur authentification 4D Mobile
C_TEXTE($1;$2)
C_BOOLEEN($0;$3)
    //$1 : utilisateur
    //$2 : mot de passe
    //$3 : mode digest
Si($1="admin")
      Si($3)
         $0:=($2=Generer digest("123";Digest 4D))
      Sinon
         $0:=($2="123")
      Fin de si
Sinon
      $0:=Faux
Fin de si
```

Exemple 2

Cet exemple de **Méthode base Sur authentification 4D Mobile** vérifie que la demande de connexion provient d'un des deux serveurs Wakanda autorisés, enregistrés dans les utilisateurs de la base 4D :

```
C_TEXTE($1;$2)
C_BOOLEEN($0)
APPELER SUR ERREUR("4DMOBILE_error")
Si($1="WAK1")|($1="WAK2")
      $0:=Valider mot de passe($1;$2;$3)
Sinon
      $0:=Faux
Fin de cas
```

Méthode base Sur authentification SQL

$1, $2, $3 -> Méthode base Sur authentification SQL -> $0

Paramètre	Type		Description
$1	Texte	⇐	Nom d'utilisateur
$2	Texte	⇐	Mot de passe
$3	Texte	⇐	(Facultatif) Adresse IP du client à l'origine de la requête
$0	Booléen	⇒	Vrai = requête acceptée, Faux = requête rejetée

La **Méthode base Sur authentification SQL** permet de filtrer les requêtes adressées au serveur SQL intégré de 4D. Le filtrage peut être effectué sur la base du nom, du mot de passe ainsi que (facultativement) de l'adresse IP de l'utilisateur. Le développeur peut utiliser sa propre table d'utilisateurs ou celle des utilisateurs 4D pour évaluer les identifiants de connexion. Une fois la connexion authentifiée, la commande **CHANGER UTILISATEUR COURANT** doit être appelée afin de contrôler les accès de la requête au sein de la base 4D.

Lorsqu'elle existe, la **Méthode base Sur authentification SQL** est automatiquement appelée par 4D ou 4D Server à chaque connexion externe au serveur SQL. Le système interne de gestion des utilisateurs de 4D n'est alors pas sollicité. La connexion n'est acceptée que si la méthode base retourne **Vrai** dans $0 et si la commande **CHANGER UTILISATEUR COURANT** a été exécutée avec succès. Si l'une des deux conditions n'est pas remplie, la requête est rejetée.

Note : L'instruction **SQL LOGIN(SQL_INTERNAL;$utilisateur;$motdepasse)** ne déclenche pas l'appel de la **Méthode base Sur authentification SQL** car il s'agit dans ce cas d'une connexion interne.

La méthode base reçoit jusqu'à trois paramètres de type Texte, passés par 4D ($1, $2 et $3), et retourne un booléen, $0. Voici la description de ces paramètres :

Paramètres	Type	Description
$1	Texte	Nom d'utilisateur
$2	Texte	Mot de passe
$3	Texte	(Facultatif) Adresse IP du client à l'origine de la requête(*)
$0	Booléen	Vrai = requête acceptée, Faux = requête rejetée

(*) 4D retourne les adresses IPv4 dans un format hybride IPv6 comprenant un préfixe de 96 bits, par exemple ::ffff:192.168.2.34 pour l'adresse IPv4 192.168.2.34. Pour plus d'informations, reportez-vous à la section **Prise en charge d'IP v6**.

Vous devez déclarer ces paramètres de la manière suivante :

```
  ` Méthode base Sur authentification Web
C_TEXTE($1;$2;$3)
C_BOOLEEN($0)
... ` Code pour la méthode
```

Le mot de passe ($2) est reçu en texte standard.

Vous devez contrôler les identifiants de la connexion SQL dans la **Méthode base Sur authentification SQL**. Par exemple, vous pouvez contrôler le nom et le mot de passe à l'aide d'une table d'utilisateurs personnalisée. Si les identifiants sont valides, passez Vrai dans $0. Sinon, passez **Faux** dans $0, dans ce cas la connexion est rejetée.

Si vous avez passé **Vrai** dans $0, vous devez ensuite appeler avec succès la commande **CHANGER UTILISATEUR COURANT** dans la **Méthode base Sur authentification SQL** pour que la requête soit acceptée et que 4D ouvre une session SQL pour l'utilisateur.
L'utilisation de la commande **CHANGER UTILISATEUR COURANT** permet de mettre en place un système d'authentification virtuelle ayant comme double avantage le contrôle des actions au sein de la connexion et le masquage pour l'extérieur des identifiants de la connexion dans la session SQL 4D.

Note : Si la **Méthode base Sur authentification SQL** n'existe pas, la connexion est évaluée à l'aide du système intégré de gestion des utilisateurs de 4D s'il est actif, c'est-à-dire si un mot de passe a été attribué au Super_Utilisateur. Si le système n'est pas actif, les utilisateurs sont connectés avec les droits du Super_Utilisateur (accès libre).

Cet exemple de **Méthode base Sur authentification SQL** vérifie que la demande de connexion provient du réseau interne, valide les identifiants puis affecte les droits d'utilisateur "sql_user" pour la session SQL.

```
C_TEXTE($1;$2;$3)
C_BOOLEEN($0)
 `$1 : utilisateur
 `$2 : mot de passe
 `{$3 : Adresse IP du client}
APPELER SUR ERREUR("SQL_error")
Si(checkInternalIP($3))
 `La méthode checkInternalIP vérifie que l'adresse IP est interne
   Si($1="victor") & ($2="hugo")
      CHANGER UTILISATEUR COURANT("sql_user";"")
      Si(OK=1)
         $0:=Vrai
      Sinon
         $0:=Faux
      Fin de si
   Sinon
      $0:=Faux
   Fin de si
Sinon
   $0:=Faux
Fin de si
```

Méthode base Sur authentification Web

$1, $2, $3, $4, $5, $6 -> Méthode base Sur authentification Web -> $0

Paramètre	Type		Description
$1	Texte	⇐	URL
$2	Texte	⇐	En-tête + Corps HTTP
$3	Texte	⇐	Adresse IP du navigateur
$4	Texte	⇐	Adresse IP appelée du serveur
$5	Texte	⇐	Nom d'utilisateur
$6	Texte	⇐	Mot de passe
$0	Booléen	⇒	Vrai = requête acceptée, Faux = requête rejetée

Description

La **Méthode base Sur authentification Web** est chargée de gérer les accès au moteur de serveur Web. Elle est automatiquement appelée par 4D ou 4D Server lorsqu'une requête d'un navigateur Web requiert l'exécution d'une méthode 4D sur le serveur (appel d'une méthode via un URL *4DACTION*, une balise *4DSCRIPT*, etc.).

La **Méthode base Sur authentification Web** reçoit six paramètres de type Texte, passés par 4D ($1, $2, $3, $4, $5 et $6), et retourne un booléen, $0. Voici la description de ces paramètres :

Paramètres	Type	Description
$1	Texte	URL
$2	Texte	En-tête + Corps HTTP (32 ko maximum)
$3	Texte	Adresse IP du navigateur
$4	Texte	Adresse IP appelée du serveur
$5	Texte	Nom d'utilisateur
$6	Texte	Mot de passe
$0	Booléen	Vrai = requête acceptée, Faux = requête rejetée

Vous devez déclarer ces paramètres de la manière suivante :

```
  ` Méthode base Sur authentification Web

C_TEXTE($1;$2;$3;$4;$5;$6)
C_BOOLEEN($0)

  ` Code pour la méthode
```

Note : Tous les paramètres de la **Méthode base Sur authentification Web** ne sont pas forcément remplis. Les informations reçues par la méthode base dépendent des options que vous avez sélectionnées dans la boîte de dialogue des Propriétés de la base. Référez-vous à la section **Sécurité des connexions**.

- **URL**
  Le premier paramètre (*$1*) est l'**URL** saisi par l'utilisateur dans la zone 'Adresse' de son navigateur Web, moins l'adresse hôte.
  Prenons l'exemple d'une connexion Intranet. Supposons que l'adresse **IP** de votre machine serveur Web 4D est *123.4.567.89*. Le tableau suivant liste les valeurs de *$1* selon l'**URL** saisi dans le navigateur Web :

URL saisi dans le navigateur Web	Valeur du paramètre $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients/Ajouter	/Clients/Ajouter
123.4.567.89/Faire_ceci/Si_OK/Faire_cela	/Faire_ceci/Si_OK/Faire_cela

- **En-tête et corps de la requête HTTP**
  Le deuxième paramètre (*$2*) est l'en-tête et le corps de la requête **HTTP** envoyée par le navigateur Web. Notez que ces informations sont passées telles quelles à la **Méthode base Sur authentification Web**. Le contenu varie en fonction du type de navigateur Web qui tente de se connecter. Si votre application exploite ces informations, il est de votre ressort d'analyser l'en-tête et le corps.

Notes :

- Pour des raisons de performances, la taille des données transitant via le paramètre $2 ne peut dépasser 32 Ko. Au-delà, elles sont tronquées par le serveur HTTP de 4D.
- Pour plus d'informations sur ce paramètre, reportez-vous à la description de la **Méthode base Sur connexion Web**.

- **Adresse IP du navigateur**
  Le troisième paramètre (*$3*) reçoit l'adresse IP de la machine du navigateur. Cette information peut vous permettre, en particulier, de distinguer les connexions Intranet des connexions Internet.
  Note : 4D retourne les adresses IPv4 dans un format hybride IPv6 comprenant un préfixe de 96 bits, par exemple ::ffff:192.168.2.34 pour l'adresse IPv4 192.168.2.34. Pour plus d'informations, reportez-vous à la section **Prise en charge d'IP v6**.

- **Adresse IP demandée du serveur**
  Le quatrième paramètre (*$4*) reçoit l'adresse IP demandée du serveur Web 4D. En effet, 4D autorise le multi-homing, permettant d'exploiter des machines disposant de plusieurs adresses IP. Pour plus d'informations sur ce point, reportez-vous à la section **Paramétrages du serveur Web**.

- **Nom d'utilisateur et Mot de passe**

Les paramètres *$5* et *$6* reçoivent le nom d'utilisateur et le mot de passe saisis par l'utilisateur dans la boîte de dialogue standard d'identification affichée par le navigateur, le cas échéant.
Cette boîte de dialogue apparaît pour chaque connexion dès qu'une option de gestion des mots de passe est cochée dans les Propriétés de la base (cf. section **Sécurité des connexions**).

Note : Si le nom d'utilisateur envoyé par le navigateur existe dans 4D, pour des raisons de confidentialité le paramètre *$6* n'est alors pas rempli (il reçoit une chaîne vide).

- **Paramètre $0**
  La **Méthode base Sur authentification Web** retourne un booléen dans $0 :
  - Si $0 est **Vrai**, la connexion est acceptée.
  - Si $0 est **Faux**, la connexion est refusée.

La **Méthode base Sur connexion Web** n'est exécutée que si la connexion est acceptée par **Sur authentification Web**.

ATTENTION : Si aucune valeur n'est passée dans *$0*, ou si *$0* n'est pas définie dans la **Méthode base Sur authentification Web**, la connexion sera considérée comme acceptée, et la **Méthode base Sur connexion Web** sera exécutée.

Notes :

- N'appelez aucun élément d'interface dans la **Méthode base Sur authentification Web** ((**ALERTE**, **DIALOGUE**, etc.), sinon son exécution sera interrompue et la connexion refusée. Il en est de même si une erreur se produit durant son traitement.
- Il est possible d'interdire l'exécution par *4DACTION* ou *4DSCRIPT* de chaque méthode projet à l'aide de l'option "Disponible via les balises HTML et URLs 4D (4DACTION...)" dans la boîte de dialogue des Propriétés des méthodes. Pour plus d'informations sur ce point, reportez-vous à la section **Sécurité des connexions**.

Appels de la Méthode base Sur authentification Web

La **Méthode base Sur authentification Web** est automatiquement appelée, quel que soit le mode, lorsqu'une requête ou un traitement nécessite l'exécution d'une méthode 4D. Elle est également appelée lorsque le serveur Web reçoit un URL statique invalide (par exemple, si la page statique demandée n'existe pas).
La **Méthode base Sur authentification Web** est donc appelée dans les cas suivants :

- lorsque 4D reçoit un URL débutant par *4DACTION/*
- lorsque 4D reçoit un URL débutant par *4DCGI/*
- lorsque 4D reçoit un URL débutant par *4DSYNC/*
- lorsque 4D reçoit un URL demandant une page statique inexistante
- lorsque 4D reçoit un URL d'accès à la racine et qu'aucune page d'accueil n'est définie dans les propriétés de la base ou via la commande **WEB FIXER PAGE ACCUEIL**
- lorsque 4D traite une balise *4DSCRIPT* dans une page semi-dynamique
- lorsque 4D traite une balise *4DLOOP* basée sur une méthode dans une page semi-dynamique

Note de compatibilité : La méthode base est également appelée lorsque 4D reçoit un URL débutant par *4DMETHOD/*. Cet URL obsolète est conservé par compatibilité uniquement.
A noter que la **Méthode base Sur authentification Web** n'est PAS appelée lorsque le serveur reçoit un URL demandant une page statique valide.

Exemple 1

Exemple de **Méthode base Sur authentification Web** en mode BASIC :

```
 `Méthode base Sur authentification Web
C_TEXTE($5;$6;$3;$4)
C_TEXTE($utilisateur;$motPasse;$IPBrowser;$IPServer)
C_BOOLEEN($utilisateur4D)
TABLEAU TEXTE($utilisateurs;0)
TABLEAU ENTIER LONG($nums;0)
C_ENTIER LONG($upos)
C_BOOLEEN($0)

$0:=Faux

$utilisateur:=$5
$motPasse:=$6
$IPBrowser:=$3
$IPServer:=$4

 `Pour des raisons de sécurité, refuser les noms qui contiennent @
Si(AvecJoker($utilisateur)|AvecJoker($motPasse))
   $0:=Faux
 `La méthode AvecJoker est décrite ci-dessous
Sinon
 `Vérifier si c'est un utilisateur 4D
   LIRE LISTE UTILISATEURS($utilisateurs;$nums)
   $upos:=Chercher dans tableau($utilisateurs;$utilisateur)
   Si($upos >0)
      $utilisateur4D:=Non(Utilisateur supprime($nums{$upos}))
   Sinon
      $utilisateur4D:=Faux
   Fin de si

   Si(Non($utilisateur4D))
```

```
   `Ce n'est pas un utilisateur défini dans 4D, chercher dans la table des utilisateurs Web
      CHERCHER([WebUsers];[WebUsers]User=$utilisateur;*)
      CHERCHER([WebUsers]; & [WebUsers]Password=$motPasse)
      $0:=(Enregistrements trouves([WebUsers])=1)
   Sinon
      $0:=Vrai
   Fin de si
Fin de si
   `Est-ce une connexion intranet ?
Si(Sous chaine($IPBrowser;1;7)#"192.100.")
   $0:=Faux
Fin de si
```

Exemple 2

Exemple de méthode base Sur authentification Web en mode DIGEST :

```
   ` Méthode base Sur authentification Web
C_TEXTE($1;$2;$5;$6;$3;$4)
C_TEXTE($utilisateur)
C_BOOLEEN($0)
$0:=Faux
$utilisateur:=$5
   `Pour des raisons de sécurité, refuser les noms qui contiennent @
Si(AvecJoker($utilisateur))
   $0:=Faux
   `La méthode AvecJoker est décrite ci-dessous
Sinon
   CHERCHER([WebUsers];[WebUsers]User=$utilisateur)
   Si(OK=1)
      $0:=Valider mot de passe digest Web($utilisateur;[WebUsers]Mdp)
   Sinon
      $0:=Faux `Utilisateur inexistant
   Fin de si
Fin de si

La Méthode projet AvecJoker est décrite ci-dessous:
```

```
   `Méthode projet AvecJoker
   `AvecJoker ( Chaine ) -> Booléen
   `AvecJoker ( Nom ) -> Contient un joker

C_ENTIER($i)
C_BOOLEEN($0)
C_TEXTE($1)
$0:=Faux
Boucle($i;1;Longueur($1))
   Si(Code de caractere(Sous chaine($1;$i;1))=Code de caractere("@"))
      $0:=Vrai
   Fin de si
Fin de boucle
```

Méthode base Sur connexion Web

$1, $2, $3, $4, $5, $6 -> Méthode base Sur connexion Web

Paramètre	Type		Description
$1	Texte	⇐	URL
$2	Texte	⇐	En-tête + corps HTTP
$3	Texte	⇐	Adresse IP du navigateur
$4	Texte	⇐	Adresse IP appelée du serveur
$5	Texte	⇐	Nom d'utilisateur
$6	Texte	⇐	Mot de passe

La **Méthode base Sur connexion Web** peut être appelée dans les cas suivants :

- le serveur Web reçoit une requête débutant par l'URL *4DCGI*.
- le serveur Web reçoit une requête invalide.

Pour plus d'informations, reportez-vous ci-dessous au paragraphe “Appels de la Méthode base Sur connexion Web”.

Note de compatibilité : La méthode base est également appelée en cas de création d'un contexte en mode contextuel (mode obsolète pouvant être utilisé dans les bases 4D converties).

La requête doit auparavant avoir été "acceptée" par la **Méthode base Sur connexion Web** (si elle existe) et le serveur Web doit être lancé.

La **Méthode base Sur connexion Web** reçoit six paramètres de type Texte, passés par 4D ($1, $2, $3, $4, $5 et $6). Voici leur description :

Paramètres	Type	Description
$1	Texte	URL
$2	Texte	En-tête + corps HTTP (32 ko maximum)
$3	Texte	Adresse IP du navigateur
$4	Texte	Adresse IP appelée du serveur
$5	Texte	Nom d'utilisateur
$6	Texte	Mot de passe

Vous devez déclarer ces six paramètres de la manière suivante :

```
  ` Méthode base Sur connexion Web

C_TEXTE($1;$2;$3;$4;$5;$6)

  ` Code pour la méthode
```

- **Données supplémentaires de l'URL**
  Le premier paramètre (*$1*) est l'**URL** saisi par l'utilisateur dans la zone 'Adresse' de son navigateur Web, moins l'adresse hôte.
  Prenons l'exemple d'une connexion Intranet. Supposons que l'adresse **IP** de votre machine serveur Web 4D est *123.4.567.89*. Le tableau suivant liste les valeurs de *$1* selon l'**URL**saisi dans le navigateur Web :

URL saisi dans le navigateur	Valeur du paramètre $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients/Ajouter	/Clients/Ajouter
123.4.567.89/Faire_ceci/Si_OK/Faire_cela	/Faire_ceci/Si_OK/Faire_cela

  Notez que vous êtes libre d'utiliser ce paramètre à votre convenance. 4D ignore simplement les valeurs passées au-delà de la partie hôte de l'URL. Par exemple, vous pouvez établir une convention dans laquelle la valeur *"/Clients/Ajouter"* signifie “aller directement à l'ajout d'un nouvel enregistrement dans la table *[Clients]*.” En fournissant aux utilisateurs Web de votre base une liste des valeurs possibles et/ou des marqueurs par défaut, vous pouvez proposer des raccourcis vers différentes parties de votre application. Ainsi, les utilisateurs Web peuvent accéder rapidement aux ressources de votre site Web sans devoir naviguer dans toute la base à chaque connexion.

ATTENTION : Pour empêcher un utilisateur d'accéder directement à une base à l'aide d'un marqueur créé pendant une session précédente, 4D intercepte tout URL correspondant à un des URLs standard de 4D.

- **En-tête et corps de la requête HTTP**
  Le deuxième paramètre (*$2*) est l'en-tête suivi du corps de la requête **HTTP** envoyée par le navigateur Web. Notez que ces informations sont passées telles quelles à votre **Méthode base Sur connexion Web**. Le contenu varie en fonction du type de navigateur Web qui tente de se connecter.
  Avec Safari sous Mac OS, vous recevrez un en-tête semblable à celui-ci :

```
GET /favicon.ico HTTP/1.1
Referer: http://123.45.67.89/4dcgi/test
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; fr-fr) AppleWebKit/523.10.3 (KHTML, like Gecko)
Version/3.0.4 Safari/523.10
Cache-Control: max-age=0
Accept: */*
Accept-Language: fr-fr
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

```
Host: 123.45.67.89
```

Avec Microsoft Internet Explorer 8 sous Windows, vous recevrez un en-tête semblable à celui-ci :

```
GET / HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg,
application/x-ms-xbap, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: fr-FR
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET
CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C)
Accept-Encoding: gzip, deflate
Host: 123.45.67.89
Connection: Keep-Alive
```

Si votre application exploite ces informations, il est de votre ressort d'analyser l'en-tête et le corps.

Note : Pour des raisons de performances, la taille de ces données ne peut dépasser 32 Ko. Au-delà, elles sont tronquées par le serveur HTTP de 4D.

- **Adresse IP du navigateur**
  Le troisième paramètre (*$3*) reçoit l'adresse IP de la machine du navigateur. Cette information peut vous permettre, en particulier, de distinguer les connexions Intranet des connexions Internet.
  Note : 4D retourne les adresses IPv4 dans un format hybride IPv6 comprenant un préfixe de 96 bits, par exemple ::ffff:192.168.2.34 pour l'adresse IPv4 192.168.2.34. Pour plus d'informations, reportez-vous à la section **Prise en charge d'IP v6**.

- **Adresse IP demandée du serveur**
  Le quatrième paramètre (*$4*) reçoit l'adresse IP demandée du serveur Web 4D. En effet, 4D autorise le multi-homing, permettant d'exploiter des machines disposant de plusieurs adresses IP. Pour plus d'informations sur ce point, reportez-vous à la section **Paramétrages du serveur Web**.

- **Nom d'utilisateur et Mot de passe**
  Les paramètres *$5* et *$6* reçoivent le nom d'utilisateur et le mot de passe saisis par l'utilisateur dans la boîte de dialogue standard d'identification affichée par le navigateur, le cas échéant.
  Cette boîte de dialogue apparaît pour chaque connexion dès que l'option **Utiliser les mots de passe** est cochée dans les Propriétés de la base (cf. section **Sécurité des connexions**).

Note : Si le nom d'utilisateur envoyé par le navigateur existe dans 4D, pour des raisons de confidentialité le paramètre *$6* n'est alors pas rempli (il reçoit une chaîne vide).

Appels de la Méthode base Sur connexion Web

La **Méthode base Sur connexion Web** peut être utilisée comme point d'entrée dans le serveur Web 4D, soit à l'aide de l'URL spécial */4DCGI*, soit à l'aide d'URLs de commande personnalisés.

Attention : L'appel d'une commande 4D affichant un élément d'interface (**DIALOGUE**, **ALERTE**...) entraîne l'arrêt du traitement de la méthode.

La **Méthode base Sur connexion Web** est donc appelée dans les cas suivants :

- lorsque 4D reçoit l'URL */4DCGI*. La méthode base est appelée avec l'URL */4DCGI/<action>* dans *$1*.
- lorsqu'une page Web appelée avec un URL du type *<chemin>/<fichier>* n'est pas trouvée. La méthode base est appelée avec l'URL (*).
- lorsqu'une page Web est appelée avec un URL du type *<chemin>/* et qu'aucune page d'accueil par défaut n'est définie. La méthode base est appelée avec l'URL (*).

(*) Dans ces cas particuliers, l'URL reçu dans *$1* ne débute pas par le caractère "/".

Méthode base Sur démarrage sauvegarde

Méthode base Sur démarrage sauvegarde -> $0

Paramètre	Type		Description
$0	Entier long		0 = sauvegarde autorisée, valeur autre que 0 = sauvegarde non autorisée

---

La **Méthode base Sur démarrage sauvegarde** est appelée à chaque fois qu'une sauvegarde de la base est sur le point d'avoir lieu (sauvegarde manuelle, sauvegarde automatique périodique ou via la commande **SAUVEGARDER**).
Tous les environnements 4D sont concernés : 4D (tous modes), 4D Server ainsi que les applications 4D compilées et fusionnées avec 4D Volume Desktop.

La **Méthode base Sur démarrage sauvegarde** permet de contrôler le déclenchement de la sauvegarde. Au sein de la méthode, vous devez retourner dans le paramètre $0 une valeur autorisant ou refusant la sauvegarde :

- si $0 = 0, vous autorisez la sauvegarde.
- si $0 # 0, vous n'autorisez pas la sauvegarde. L'opération est annulée et une erreur est retournée. Vous pouvez récupérer l'erreur à l'aide de la commande **LIRE INFORMATION SAUVEGARDE**.

Vous pouvez utiliser cette méthode base pour contrôler les conditions d'exécution de la sauvegarde (utilisateur, date de la dernière sauvegarde, etc.).

Note : Vous devez impérativement déclarer le paramètre *$0* (entier long) dans la méthode base :

```
C_ENTIER LONG($0)
```

Méthode base Sur démarrage serveur

Méthode base Sur démarrage serveur
Ne requiert pas de paramètre

La méthode base Sur démarrage serveur est appelée une fois sur le poste serveur lorsque vous ouvrez une base avec 4D Server. La méthode base Sur démarrage serveur n'est jamais exécutée dans un environnement autre que 4D Server.

La méthode base Sur démarrage serveur est l'emplacement idéal pour :

- Initialiser les variables interprocess utilisées pendant toute la session 4D Server.
- Démarrer automatiquement des **Procédures stockées** à l'ouverture de la base.
- Charger des préférences ou des paramétrages sauvegardé(e)s dans ce but lors de la précédente session 4D Server.
- Empêcher l'ouverture de la base si une condition n'est pas remplie (par exemple, absence de ressources système) par un appel explicite à **QUITTER 4D**.
- Placer toute action devant être automatiquement effectuée à chaque ouverture de la base.

Si vous souhaitez exécuter du code automatiquement sur un poste client lorsqu'un 4D distant se connecte au serveur, utilisez la **Méthode base Sur ouverture**.

Note : La méthode base Sur démarrage serveur est exécutée de façon atomique, ce qui signifie qu'aucun 4D distant ne peut se connecter tant que l'exécution de la méthode n'est pas terminée.

Méthode base Sur déposer

Méthode base Sur déposer
Ne requiert pas de paramètre

La **Méthode base Sur déposer** est disponible dans les applications 4D locales ou distantes.

Cette méthode base est exécutée automatiquement en cas de déposer d'objets dans l'application 4D en-dehors de tout formulaire ou fenêtre, c'est-à-dire :

- dans une zone vide de la fenêtre MDI (Windows),
- sur l'icône 4D dans le Dock (Mac OS) ou sur le Bureau du système.

Sous Mac OS, il est nécessaire de maintenir les touches **Option**+**Commande** enfoncées pendant le déposer afin que la méthode base soit appelée.

Lors d'un déposer sur l'icône de l'application 4D sur le bureau, la **Méthode base Sur déposer** est appelée uniquement si l'application est déjà lancée, sauf dans le cas des applications fusionnées avec 4D Desktop. Dans ce cas, la méthode base est appelée même lorsqu'elle n'est pas lancée. Ce principe permet de définir des signatures de documents personnalisées.

Exemple

Cet exemple permet d'ouvrir un document 4D Write déposé en-dehors de tout formulaire :

```
  `Méthode base Sur Déposer
fichierDéposé:=Lire fichier dans conteneur(1)
Si(Position(".4W7";fichierDéposé)=Longueur(fichierDéposé)-3)
   zexterne:=Creer fenetre externe(100;100;500;500;0;fichierDéposé;"_4D Write")
   WR OUVRIR DOCUMENT(zexterne;fichierDéposé)
Fin de si
```

Méthode base Sur événement base hôte

$1 -> Méthode base Sur événement base hôte

Paramètre	Type		Description
$1	Entier long	⇦	Code d'événement

Description

La **Méthode base Sur événement base hôte** permet aux composants 4D d'exécuter du code lors de l'ouverture et de la fermeture de la base hôte.

Note : Pour des raisons de sécurité, l'exécution de cette méthode base doit être autorisée explicitement dans la base hôte pour qu'elle puisse être appelée. Pour plus d'informations sur ce point, reportez-vous au manuel *Mode Développement*.

La **Méthode base Sur événement base hôte** est exécutée uniquement dans les bases utilisées en tant que composants de bases hôtes (lorsqu'elle est autorisée dans les Propriétés de la base hôte). Elle est appelée lorsque des événements liés à l'ouverture et la fermeture de la base hôte se produisent.

Pour traiter un événement, vous devez tester la valeur du paramètre *$1* à l'intérieur de la méthode, et la comparer à l'une des constantes suivantes, placées dans le thème **Evénements de la base** :

Constante	Type	Valeur	Comment
Sur après fermeture base hôte	Entier long	4	La **Méthode base Sur fermeture** de la base hôte vient de terminer son exécution
Sur après ouverture base hôte	Entier long	2	La **Méthode base Sur ouverture** de la base hôte vient de terminer son exécution
Sur avant fermeture base hôte	Entier long	3	La base hôte est en cours de fermeture. La **Méthode base Sur fermeture** de la base hôte n'a pas encore été appelée. La **Méthode base Sur fermeture** de la base hôte n'est pas appelée tant que la **Méthode base Sur événement base hôte** du composant est en exécution
Sur avant ouverture base hôte	Entier long	1	La base hôte vient juste d'être lancée. La **Méthode base Sur ouverture** de la base hôte n'a pas encore été appelée. La **Méthode base Sur ouverture** de la base hôte n'est pas appelée tant que la **Méthode base Sur événement base hôte** du composant est en exécution

Ce principe permet aux composants 4D de charger et de sauvegarder des préférences ou des états utilisateurs liés à l'exploitation de la base hôte.

Exemple

Exemple de structure type d'une méthode base sur événement base hôte :

```
    // Méthode base sur événement base hôte
C_ENTIER LONG($1)
Au cas ou
         :($1=Sur avant ouverture base hôte)
            // placer ici le code à exécuter avant le "Sur ouverture" de la base hôte

         :($1=Sur après ouverture base hôte)
            // placer ici le code à exécuter après le "Sur ouverture" de la base hôte

         :($1=Sur avant fermeture base hôte)
            // placer ici le code à exécuter avant le "Sur fermeture" de la base hôte

         :($1=Sur après fermeture base hôte)
            // placer ici le code à exécuter après le "Sur fermeture" de la base hôte
Fin de cas
```

Méthode base Sur événement système

$1 -> Méthode base Sur événement système

Paramètre	Type		Description
$1	Entier long	⇐	Code d'événement

Description

La **Méthode base Sur événement système** est appelée à chaque fois qu'un événement système se produit. Tous les environnements 4D sont concernés : 4D (tous modes), 4D Server ainsi que les applications 4D compilées et fusionnées avec 4D Volume Desktop.

Pour traiter un événement, vous devez tester la valeur du paramètre $1 à l'intérieur de la méthode, et la comparer à l'une des constantes suivantes, placées dans le thème **Evénements de la base** :

Constante	Type	Valeur	Comment
Sur passage arrière plan	Entier long	1	L'application 4D passe à l'arrière plan
Sur passage premier plan	Entier long	2	L'application 4D passe au premier plan

Ces événements sont générés lorsque l'application 4D change de plan, quelle que soit l'action utilisateur à l'origine du changement :

- clic dans la fenêtre de l'application ou d'une autre application,
- sélection via le raccourci clavier **Alt+Tab** (Windows) ou **Commande+Tab** (Mac OS),
- sélection de la commande **Masquer** dans le dock (Mac OS),
- clic sur l'icône de l'application dans le dock ou la barre des tâches,
- clic sur le bouton de réduction de la fenêtre principale (Windows).

Vous devez impérativement déclarer le paramètre $1 (entier long) dans la méthode base. La structure du code de la méthode base sera donc :

```
 // Méthode base Sur événement système

C_ENTIER LONG($1)
Au cas ou
   :($1=Sur passage arrière plan)
 //Faire quelque chose
   :($1=Sur passage premier plan)
 //Faire autre chose
Fin de cas
```

Méthode base Sur fermeture

Méthode base Sur fermeture
Ne requiert pas de paramètre

La **Méthode base Sur fermeture** est appelée une fois lorsque vous quittez la base.

Les environnements 4D suivants sont concernés :

- 4D en mode local
- 4D en mode distant
- Application 4D compilée et fusionnée avec 4D VolumeDesktop

Note : La **Méthode base Sur fermeture** n'est PAS exécutée par 4D Server.

La **Méthode base Sur fermeture** est exécutée automatiquement par 4D ; à la différence des méthodes projet, vous ne pouvez pas appeler cette méthode base par programmation. Vous pouvez toutefois l'exécuter depuis l'éditeur de méthodes. Vous pouvez également utiliser des sous-routines.

On sort de la base lorsque l'un des événements suivants se produit :

- L'utilisateur sélectionne la commande **Quitter** dans le menu **Fichier** en mode Développement ou Application (action standard Quitter).
- Un appel à la commande **QUITTER 4D** a eu lieu.
- Un plug-in 4D a fait appel au point d'entrée **QUITTER 4D**.

Quel que soit le moyen par lequel la base a été quittée, 4D accomplit les actions qui suivent :

- S'il n'existe pas de **Méthode base Sur fermeture**, 4D détruit chaque process un par un, sans distinction. Si un utilisateur est en train de saisir des données, les enregistrements ne seront pas sauvegardés.
- S'il existe une **Méthode base Sur fermeture**, 4D démarre l'exécution de cette méthode dans un process local nouvellement créé. Vous pouvez ainsi utiliser cette méthode base pour informer d'autres process, via la communication interprocess, qu'ils doivent être fermés (en cas de saisie de données) ou stopper leur exécution. Notez que 4D quittera en tout état de cause — la **Méthode base Sur fermeture** peut assurer le nettoyage et la fermeture de toutes les opérations que vous voulez, mais la fermeture de la base est inéluctable.

La **Méthode base Sur fermeture** est l'emplacement idéal pour :

- Stopper les process automatiquement démarrés à l'ouverture de la base.
- Sauvegarder (localement, sur disque) les préferences ou paramétrages devant être réutilisés lors de la prochaine session dans la **Méthode base Sur ouverture**.
- Accomplir toute autre action que vous souhaitez exécuter automatiquement à chaque fermeture de la base.

Note: Rappelez-vous que le process créé pour la **Méthode base Sur fermeture** est un process client (local), qui n'existe donc pas sur le poste serveur. Par conséquent, si vous effectuez dans cette méthode base une recherche ou un tri, tout poste client qui tentera de quitter l'application restera "bloqué". Si vous avez besoin d'accéder aux données lorsque le client quitte, vous devez créer depuis cette méthode base un process global qui, lui, pourra accéder aux données. Dans ce cas toutefois, veillez à ce que ce process puisse terminer son exécution (par l'intermédaire de variables interprocess, par exemple) avant d'être stoppé par la **Méthode base Sur fermeture**.

Exemple

L'exemple ci-dessous liste les méthodes utilisées dans une base qui note les événements significatifs se produisant lors d'une session de travail. Les étapes sont écrites dans un document texte appelé “Journal”.

- La **Méthode base Sur ouverture** initialise la variable interprocess *◊vbQuit4D*, qui signale tous les process utilisés, qu'on sorte ou non de la base. Elle crée aussi le fichier journal, s'il n'existe pas déjà.

```
  ` Méthode base Sur ouverture
C_TEXTE(◊vtIPMessage)
C_BOOLEEN(◊vbQuit4D)
◊vbQuit4D:=Faux

Si(Tester chemin acces("Journal") # Est un document)
   $vhDocRef:=Creer document("Journal")
   Si(OK=1)
      FERMER DOCUMENT($vhDocRef)
   Fin de si
Fin de si
ECRIRE JOURNAL("Ouverture Session")
```

- La méthode projet ***ECRIRE JOURNAL***, utilisée comme sous-routine par les autres méthodes, écrit l'information qu'elle reçoit dans le fichier journal :

```
  ` Méthode Projet ECRIRE JOURNAL
  ` ECRIRE JOURNAL ( Texte )
  ` ECRIRE JOURNAL ( Description Evenement )
C_TEXTE($1)
C_HEURE($vhDocRef)
```

```
Tant que(Semaphore("$Journal"))
   ENDORMIR PROCESS(Numero du process courant;1)
Fin tant que
$vhDocRef:=Ajouter a document("Journal")
Si(OK=1)
   INFORMATIONS PROCESS(Numero du process courant;$vsProcessNom;$vlEtat;$vlTempsEcoule;$vbVisible)
   ENVOYER PAQUET($vhDocRef;Chaine(Date du jour)+Caractere(9)+Chaine(Heure courante)+Caractere(9)+Chaine(Numero du
process courant)+Caractere(9)+$vsProcessNom+Caractere(9)+$1+Caractere(13))
   FERMER DOCUMENT($vhDocRef)
Fin de si
EFFACER SEMAPHORE("$Journal")
```

Notez que le document est ouvert et refermé à chaque fois. Notez aussi l'emploi d'un sémaphore comme “protection d'accès” au document — nous ne voulons pas que deux process essaient d'accéder au fichier journal en même temps.

- La méthode projet ***M_AJOUT_ENRG*** est exécutée lorsque la commande de menu **Ajouter enregistrement** est sélectionnée en mode Application :

```
` Méthode Projet M_AJOUT_ENRG

CHANGER BARRE(1)
Repeter
   AJOUTER ENREGISTREMENT([Table1];*)
   Si(OK=1)
      ECRIRE JOURNAL("Ajout d'enregistrement #"+Chaine(Numero enregistrement([Table1]))+" dans Table1")
   Fin de si
Jusque((OK=0) | ◊vbQuit4D)
```

Cette méthode effectue une boucle jusqu'à ce que l'utilisateur annule la saisie de données ou que la base soit refermée.

- Le formulaire entrée de la *[Table1]* inclut le traitement des événements Sur appel extérieur. Ainsi, même si un process est en saisie de données, on en sort "en douceur", et l'utilisateur peut sauvegarder (ou non) la saisie en cours :

```
` Méthode formulaire [Table1];"Entrée"
Au cas ou
   :(Evenement formulaire=Sur appel extérieur)
      Si(◊vtIPMessage="QUITTER")
         CONFIRMER("Voulez-vous sauvegarder les modifications dans cet enregistrement ?")
         Si(OK=1)
            VALIDER
         Sinon
            NE PAS VALIDER
         Fin de si
      Fin de si
Fin de cas
```

- La méthode projet ***M_QUIT*** est exécutée lorsque la commande **Quitter** du menu **Fichier** en mode Application est sélectionnée :

```
` Méthode Projet M_QUIT
$vlProcessID:=Nouveau process("ON_QUIT";32*1024;"$ON_QUIT")
```

Cette méthode utilise une astuce. Lorsque la commande **QUITTER 4D** est appelée, elle a un effet immédiat. En conséquence, le process dans lequel elle est appelée est placé en “mode arrêt”, jusqu'à ce que la base ait été effectivement refermée. Comme ce process peut être un des process dans lequel est effectuée la saisie de données, l'appel à **QUITTER 4D** est réalisé dans un process local qui n'est démarré que pour ce but. Voici la méthode ***ON_QUIT***:

```
` Méthode projet ON_QUIT
CONFIRMER("Etes-vous certain de vouloir quitter ?")
Si(OK=1)
   ECRIRE JOURNAL("Sortie de la base")
   QUITTER 4D
   ` QUITTER 4D a un effet immédiat. Aucune ligne de code n'est exécutée par la suite.
   ` ...
Fin de si
```

- Enfin, voici la **Méthode base Sur fermeture**, qui signale à tous les process utilisateur qu'“il est temps de partir !”. Elle met *◊vbQuit4D* à Vrai et envoie des messages interprocess aux process utilisateur qui gèrent la saisie de données :

```
` Méthode base Sur fermeture
◊vbQuit4D:=Vrai
Repeter
   $vbfini:=Vrai
   Boucle($vlProcess;1;Nombre de process)
      INFORMATIONS PROCESS($vlProcess;$vsProcessNom;$vlEtat;$vlTempsEcoule;$vbVisible)
      Si((((($vsProcessNom="ML_@") | ($vsProcessNom="M_@"))) & ($vlEtat>=0))
         $vbfini:=Faux
         ◊vtIPMessage:="QUITTER"
         PASSER AU PREMIER PLAN($vlProcess)
```

```
            APPELER PROCESS($vlProcess)
            $vhStart:=Heure courante
            Repeter
               ENDORMIR PROCESS(Numero du process courant;60)
            Jusque((Statut du process($vlProcess)<0) | ((Heure courante-$vhStart)>=?00:01:00?))
         Fin de si
      Fin de boucle
   Jusque($vbfini)
   ECRIRE JOURNAL("Fermeture de session")
```

Note : Les process dont les noms commencent par "**ML_...**" ou "**M_...**" sont démarrés par les commandes de menus pour lesquelles la propriété **Démarrer un process** a été sélectionnée. Dans cet exemple, ce sont les process démarrés suite à la sélection de la commande de menu **Ajouter enregistrement**.

Le test *(Heure courante-$vhStart)>=?00:01:00?* permet à la méthode base de sortir de la boucle “en attente de l'autre process”, si l'autre process ne réagit pas pendant une minute.

- Voici un exemple type de fichier Journal produit par la base :

2/6/03	15:47:25	1	Process principal	Ouverture de Session
2/6/03	15:55:43	5	ML_1	Ajout d'enregistrement #23 dans Table1
2/6/03	15:55:46	5	ML_1	Ajout d'enregistrement #24 dans Table1
2/6/03	15:55:54	6	$On_QUIT	Sortie de la base
2/6/03	15:55:58	7	$xx	Fermeture de session

Note : *$xx* est le nom du process local démarré par 4D pour exécuter la **Méthode base Sur fermeture**.

Méthode base Sur fermeture connexion serveur

$1, $2, $3 -> Méthode base Sur fermeture connexion serveur

Paramètre	Type		Description
$1	Entier long	⇦	Numéro d'utilisateur utilisé en interne par 4D Server pour identifier les utilisateurs
$2	Entier long	⇦	Numéro de connexion utilisé en interne par 4D Server pour identifier une connexion
$3	Entier long	⇦	Obsolète : Retourne toujours 0 mais doit être déclaré

Description

La **Méthode base Sur fermeture connexion serveur** est exécutée sur le poste serveur à chaque fois qu'un process 4D Client est refermé.

Comme pour la **Méthode base Sur ouverture connexion serveur**, 4D Server passe trois paramètres de type Entier long à la **Méthode base Sur fermeture connexion serveur**. En revanche, 4D Server n'attend pas de résultat en retour.

Par conséquent, la méthode doit contenir la déclaration explicite des trois paramètres Entier long :

```
C_ENTIER LONG($1;$2;$3)
```

Le tableau suivant détaille les informations fournies par les trois paramètres passés à la méthode base :

Paramètre	Description
$1	Numéro d'utilisateur utilisé en interne par 4D Server pour identifier les utilisateurs
$2	Numéro de connexion utilisé en interne par 4D Server pour identifier une connexion
$3	Obsolète : Retourne toujours 0 mais doit être déclaré

La **Méthode base Sur fermeture connexion serveur** est le pendant inverse de la **Méthode base Sur ouverture connexion serveur**. Pour plus d'informations sur ce point, ainsi que pour la description des **process 4D Client**, reportez-vous à la description de cette méthode base.

Exemple

Reportez-vous au premier exemple de la **Méthode base Sur ouverture connexion serveur**.

Méthode base Sur fermeture process Web

Méthode base Sur fermeture process Web
Ne requiert pas de paramètre

La **Méthode base Sur fermeture process Web** est appelée par le serveur Web de 4D à chaque fois qu'une session Web est sur le point d'être refermée. Une session peut être refermée dans les cas suivants :

- lorsque le nombre maximum de sessions simultanées est atteint (100 par défaut, modifiable via la commande **WEB FIXER OPTION**), et que 4D a besoin d'en créer de nouvelles (4D détruit automatiquement le process de la session inactive la plus ancienne),
- lorsque la période maximale d'inactivité du process de la session est atteinte (480 minutes par défaut, modifiable via la commande **WEB FIXER OPTION**),
- lorsque la commande **WEB FERMER SESSION** est appelée.

Au moment de l'appel de la méthode base, le contexte de la session (variables et sélections générées par l'utilisateur) est toujours valide. Ce principe vous permet donc de stocker les données relatives à la session afin de pouvoir les réutiliser par la suite, en particulier via la **Méthode base Sur connexion Web**.

Note : Dans le contexte d'une session 4D Mobile (pouvant générer plusieurs process), la **Méthode base Sur fermeture process Web** est appelée pour chaque process Web refermé, vous permettant de sauvegarder tout type de donnée (variable, sélection, etc.) générée par le process de session 4D Mobile.

Un exemple d'utilisation de la **Méthode base Sur fermeture process Web** est fourni dans la section **Gestion des sessions Web**.

Méthode base Sur ouverture

Méthode base Sur ouverture
Ne requiert pas de paramètre

La **Méthode base Sur ouverture** est exécutée une seule fois, au moment de l'ouverture de la base.

Les environnements 4D suivants sont concernés :

- 4D en mode local
- 4D en mode distant (sur le poste client une fois que la connexion a été acceptée par 4D Server)
- Application 4D compilée et fusionnée avec 4D VolumeDesktop

Note : La **Méthode base Sur ouverture** n'est PAS exécutée par 4D Server.

La **Méthode base Sur ouverture** est exécutée automatiquement par 4D ; à la différence des méthodes projet, vous ne pouvez pas appeler cette méthode base par programmation. Vous pouvez toutefois l'exécuter depuis l'éditeur de méthodes. Vous pouvez également utiliser des sous-routines.

La **Méthode base Sur ouverture** est l'emplacement idéal pour :

- Initialiser les variables interprocess que vous utiliserez pendant toute la session de travail.
- Démarrer automatiquement des process à l'ouverture de la base.
- Charger des préférences ou des paramétrages sauvegardés dans ce but lors de la session de travail précédente.
- Empêcher l'ouverture de la base si une condition n'est pas remplie (comme par exemple, une ressource système manquante) par l'appel explicite de la commande **QUITTER 4D**.
- Accomplir toute autre action que vous souhaitez exécuter automatiquement à chaque ouverture de la base.

En revanche, il est fortement déconseillé de lancer des impressions depuis la **Méthode base Sur ouverture**.

Exemple

Reportez-vous à l'exemple de la section **Méthode base Sur fermeture**.

Méthode base Sur ouverture connexion serveur

$1, $2, $3 -> Méthode base Sur ouverture connexion serveur -> $0

Paramètre	Type		Description
$1	Entier long	⇦	Numéro d'utilisateur utilisé en interne par 4D Server pour identifier les utilisateurs
$2	Entier long	⇦	Numéro de connexion utilisé en interne par 4D Server pour identifier une connexion
$3	Entier long	⇦	Obsolète : Retourne toujours 0 (mais doit être déclaré)
$0	Entier long	⇨	0 ou omis = connexion acceptée, autre valeur = connexion refusée

Quand la Méthode base Sur ouverture connexion serveur est-elle appelée ?

La **Méthode base Sur ouverture connexion serveur** est appelée une fois sur la machine serveur chaque fois qu'un poste 4D distant démarre un process de connexion. La **Méthode base Sur ouverture connexion serveur** n'est appelée que par 4D Server, à l'exclusion de tout autre environnement 4D.

La **Méthode base Sur ouverture connexion serveur** est appelée à chaque fois que :

- un 4D distant se connecte (démarrage du process principal)
- un 4D distant ouvre l'environnement Développement (démarrage du process de développement)
- un 4D distant démarre un process global (dont le nom de ne commence pas par "$") qui nécessite la création d'un process coopératif sur le serveur **(*)**. Ce process peut être créé avec la commande **Nouveau process**, une commande de menu ou la boîte de dialogue "Exécuter une méthode" .

Dans chaque cas, plusieurs process démarrent. Un sur la machine client, et un ou deux autres sur la machine serveur (suivant les besoins). Sur la machine client, le process exécute le code et envoie les requêtes à 4D Server. Sur la machine serveur, le **process 4D Client** (process préemptif) gère l'environnement de base de données du process client (c.-à-d. les sélections courantes et le verrouillage des enregistrements pour le process utilisateur) et répond aux requêtes envoyées par le process exécuté sur la machine cliente. Le **process base 4D Client** (process coopératif) est chargé de contrôler le process 4D Client correspondant.

(*) A compter de 4D v13, pour des raisons d'optimisation les process serveurs (process préemptif pour les accès au moteur de la base et process coopératif pour l'accès au langage) ne sont créés qu'en cas de nécessité lors de l'exécution du code côté client. Par exemple, voici le détail d'une séquence de code 4D s'exécutant dans un nouveau process client :

```
 // le process global commence sans nouveau process sur le serveur, comme un process local.
CREER ENREGISTREMENT([Table_1])
[Table_1])champ1_1:="Hello world"
STOCKER ENREGISTREMENT([Table_1]) // création ici du process préemptif sur le serveur
 //pas d'appel de Sur ouverture connexion serveur
$serverTime:=Heure courante(*) // création ici du process coopératif sur le serveur
 // appel de Sur ouverture connexion serveur
```

Important : Les connexions Web et les connexions SQL ne provoquent **pas** l'exécution de la **Méthode base Sur ouverture connexion serveur**. Lorsqu'un navigateur Web se connecte à 4D Server, la **Méthode base Sur authentification Web** (si elle existe) et/ou la **Méthode base Sur connexion Web** sont appelées. Lorsque 4D Server reçoit une requête SQL, la **Méthode base Sur authentification SQL** (si elle existe) est appelée. Pour plus d'informations, reportez-vous à la description de ces méthodes base dans le manuel Langage de 4D.

Important : Lors du démarrage d'une procédure stockée, la **Méthode base Sur ouverture connexion serveur** n'est **pas** appelée. Les **Procédures stockées** sont des process serveur et non des process 4D Client. Elles exécutent du code sur la machine serveur mais ne répondent pas aux requêtes échangées par 4D Client (ou d'autres clients) et 4D Server.

Comment la méthode base est-elle appelée ?

La **Méthode base Sur ouverture connexion serveur** est exécutée sur le poste serveur dans le process 4D Client qui a provoqué l'appel de la méthode.

Si, par exemple, un 4D distant se connecte à une base 4D Server en mode interprété, il démarre le process utilisateur, le process de développement ainsi que (par défaut) le process d'inscription du client. La **Méthode base Sur ouverture connexion serveur** est donc exécutée trois fois de suite. La première fois dans le process principal, la deuxième fois dans le process d'inscription du client et la troisième fois dans le process de développement. Si les trois process sont respectivement les 6e, 7e et 8e process démarrés sur la machine serveur, et si vous appelez **Numero du process courant** dans la **Méthode base Sur ouverture connexion serveur**, le premier **Numero du process courant** retourne 6, le deuxième 7 et le troisième 8.

Notez que la **Méthode base Sur ouverture connexion serveur** s'exécute sur le poste serveur, à l'intérieur du process 4D Client sur le serveur. Elle ignore tout du process exécuté sur le client. En outre, au moment où la méthode est appelée, le process 4D Client n'est pas encore nommé (**INFORMATIONS PROCESS** ne retournera pas, à ce moment, le nom du process 4D Client).

La **Méthode base Sur ouverture connexion serveur** n'a pas accès à la table des variables process du process exécuté sur le client. Cette table réside sur le poste client, pas sur le serveur.

Lorsque la **Méthode base Sur ouverture connexion serveur** accède à une variable process, elle travaille avec une table de variables process particulière, créée dynamiquement pour le process 4D Client.

4D Server passe trois paramètres de type Entier long à la **Méthode base Sur ouverture connexion serveur** et attend un résultat Entier long. La méthode doit donc être explicitement déclarée avec trois paramètres Entier long ainsi qu'un retour de fonction Entier long :

```
C_ENTIER LONG($0;$1;$2;$3)
```

Si vous ne retournez pas de valeur dans *$0* et donc laissez la variable indéfinie ou initialisée à zéro, 4D Server estime que la méthode base accepte la connexion. Si vous n'acceptez pas la connexion, retournez une valeur non nulle dans *$0*.

Le tableau ci-dessous détaille les informations fournies par les trois paramètres passés à la méthode base :

Paramètre	Description
$1	Numéro d'utilisateur utilisé en interne par 4D Server pour identifier les utilisateurs
$2	Numéro de connexion utilisé en interne par 4D Server pour identifier une connexion
$3	Obsolète : Retourne toujours 0 (mais doit être déclaré)

Ces numéros de référence ne sont pas directement utilisables en tant que « sources d'information » à passer, par exemple, comme paramètres à une

commande 4D. Ils vous fournissent un moyen d'identifier de manière unique un process 4D Client entre la **Méthode base Sur ouverture connexion serveur** et la **Méthode base Sur fermeture connexion serveur**. La combinaison de ces valeurs est unique à tout moment d'une session 4D Server. Si vous stockez cette information dans une table ou un tableau interprocess, les deux méthodes base peuvent échanger des informations. Dans l'exemple présenté à la fin de cette section, les deux méthodes base utilisent cette information pour stocker l'heure et la date du début et de la fin d'une connexion dans le même enregistrement d'une table.

Exemple 1

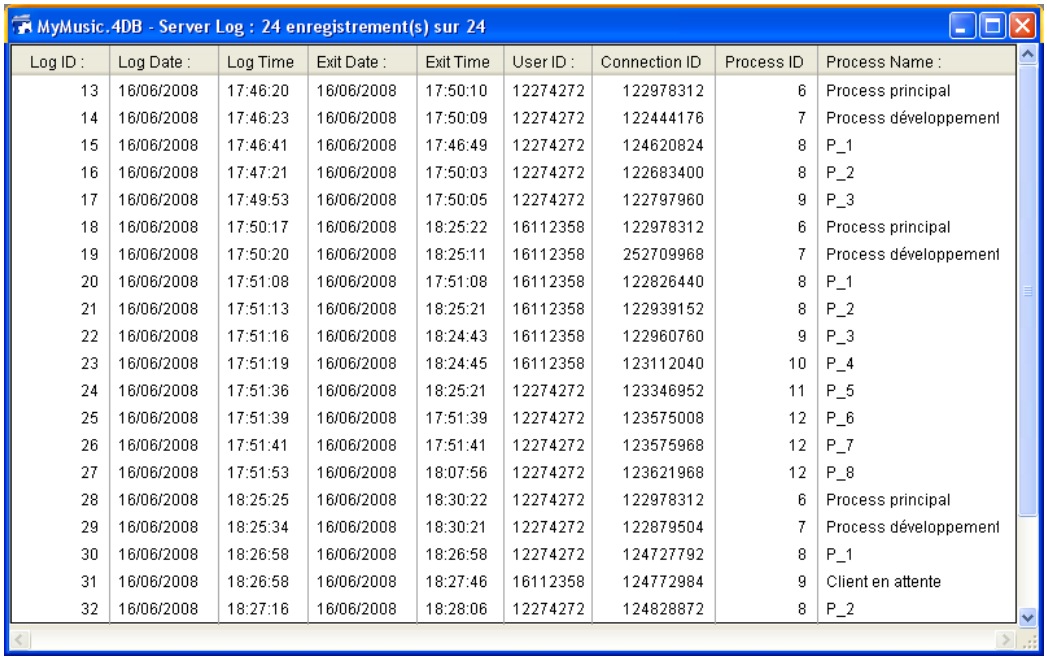
L'exemple suivant montre comment maintenir un historique des connexions à la base de données en utilisant la **Méthode base Sur ouverture connexion serveur** et la **Méthode base Sur fermeture connexion serveur**. La table [Server Log] (ci-dessous) sert à garder la trace des process de connexion :

L'information stockée dans cette table est gérée par la **Méthode base Sur ouverture connexion serveur** et la **Méthode base Sur fermeture connexion serveur** listées ci-dessous :

```
  ` Méthode base Sur ouverture connexion serveur
C_ENTIER LONG($0;$1;$2;$3)
  ` Créer un enregistrement [Server Log]
CREER ENREGISTREMENT([Server Log])
[Server Log]Log ID:=Numerotation automatique([Server Log])
  ` Enregistrer l'historique Date et Heure
[Server Log]Log Date:=Date du jour
[Server Log]Log Time:=Heure courante
  ` Enregistrer l'information sur la connexion
[Server Log]User ID:=$1
[Server Log]Connection ID:=$2
STOCKER ENREGISTREMENT([Server Log])
   ` Ne retourne pas d'erreur, pour continuer la connexion
$0:=0
```

```
  ` Méthode base Sur fermeture connexion serveur
C_ENTIER LONG($1;$2;$3)
  ` Chercher l'enregistrement [Server Log]
CHERCHER([Server Log];[Server Log]User ID=$1;*)
CHERCHER([Server Log]; & ;[Server Log]Connection ID=$2;*)
CHERCHER([Server Log]; & ;[Server Log]Process ID=0)
  ` Enregistrer date et heure de déconnexion
[Server Log]Exit Date:=Date du jour
[Server Log]Exit Time:=Heure courante
  ` Enregistrer informations process
[Server Log]Process ID:=Numero du process courant
INFORMATIONS PROCESS([Server Log]Process ID;$vsProcName;$vlProcState;$vlProcTime)
[Server Log]Process Name:=$vsProcName
STOCKER ENREGISTREMENT([Server Log])
```

Voici quelques entrées dans [Server Log] montrant plusieurs connexions distantes :



MyMusic.4DB - Server Log : 24 enregistrement(s) sur 24

Log ID :	Log Date :	Log Time	Exit Date :	Exit Time	User ID :	Connection ID	Process ID	Process Name :
13	16/06/2008	17:46:20	16/06/2008	17:50:10	12274272	122978312	6	Process principal
14	16/06/2008	17:46:23	16/06/2008	17:50:09	12274272	122444176	7	Process développement
15	16/06/2008	17:46:41	16/06/2008	17:46:49	12274272	124620824	8	P_1
16	16/06/2008	17:47:21	16/06/2008	17:50:03	12274272	122683400	8	P_2
17	16/06/2008	17:49:53	16/06/2008	17:50:05	12274272	122797960	9	P_3
18	16/06/2008	17:50:17	16/06/2008	18:25:22	16112358	122978312	6	Process principal
19	16/06/2008	17:50:20	16/06/2008	18:25:11	16112358	252709968	7	Process développement
20	16/06/2008	17:51:08	16/06/2008	17:51:08	16112358	122826440	8	P_1
21	16/06/2008	17:51:13	16/06/2008	18:25:21	16112358	122939152	8	P_2
22	16/06/2008	17:51:16	16/06/2008	18:24:43	16112358	122960760	9	P_3
23	16/06/2008	17:51:19	16/06/2008	18:24:45	16112358	123112040	10	P_4
24	16/06/2008	17:51:36	16/06/2008	18:25:21	12274272	123346952	11	P_5
25	16/06/2008	17:51:39	16/06/2008	17:51:39	12274272	123575008	12	P_6
26	16/06/2008	17:51:41	16/06/2008	17:51:41	12274272	123575968	12	P_7
27	16/06/2008	17:51:53	16/06/2008	18:07:56	12274272	123621968	12	P_8
28	16/06/2008	18:25:25	16/06/2008	18:30:22	12274272	122978312	6	Process principal
29	16/06/2008	18:25:34	16/06/2008	18:30:21	12274272	122879504	7	Process développement
30	16/06/2008	18:26:58	16/06/2008	18:26:58	12274272	124727792	8	P_1
31	16/06/2008	18:26:58	16/06/2008	18:27:46	16112358	124772984	9	Client en attente
32	16/06/2008	18:27:16	16/06/2008	18:28:06	12274272	124828872	8	P_2

Exemple 2

L'exemple suivant interdit toute nouvelle connexion entre 2 et 4 heures du matin.

```
  ` Méthode base Sur ouverture connexion serveur
C_ENTIER LONG($0;$1;$2;$3)

Si((?02:00:00?<=Heure courante)&(Heure courante<?04:00:00?))
   $0:=22000
Sinon
   $0:=0
Fin de si
```

Objets (Formulaires)

Zones 4D Write Pro

La plupart des commandes du thème "**Objets (Formulaires)**" prennent en charge les zones 4D Write Pro. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser les commandes du thème Objets (Formulaires)** dans le guide de référence de 4D Write Pro.

- Objets de formulaires
- LIRE INFOS FEUILLE DE STYLE
- LISTE FEUILLES DE STYLE
- OBJET DEPLACER
- OBJET DUPLIQUER
- OBJET Est un texte style
- OBJET FIXER ACTION
- OBJET FIXER ACTIVATION
- OBJET FIXER ALIGNEMENT HORIZONTAL
- OBJET FIXER ALIGNEMENT VERTICAL
- OBJET FIXER BARRES DEFILEMENT
- OBJET FIXER CASE A COCHER TROIS ETATS
- OBJET FIXER CONFIGURATION CLAVIER
- OBJET FIXER COORDONNEES
- OBJET FIXER CORRECTION ORTHOGRAPHIQUE
- OBJET FIXER COULEUR
- OBJET FIXER COULEURS RVB
- OBJET FIXER DEFILEMENT
- OBJET FIXER EQUIVALENT CLAVIER
- OBJET FIXER EVENEMENTS
- OBJET FIXER FEUILLE DE STYLE
- OBJET FIXER FILTRE SAISIE
- OBJET FIXER FORMATAGE
- OBJET FIXER IMPRESSION TAILLE VARIABLE
- OBJET FIXER LISTE PAR NOM
- OBJET FIXER LISTE PAR REFERENCE
- OBJET FIXER MENU CONTEXTUEL
- OBJET FIXER MESSAGE AIDE
- OBJET FIXER MULTILIGNE
- OBJET FIXER OPTIONS GLISSER DEPOSER
- OBJET FIXER ORIENTATION TEXTE
- OBJET FIXER POLICE
- OBJET FIXER RAYON ARRONDI
- OBJET FIXER RECTANGLE FOCUS INVISIBLE
- OBJET FIXER REDIMENSIONNEMENT
- OBJET FIXER SAISISSABLE
- OBJET FIXER SOURCE DONNEES
- OBJET FIXER SOUS FORMULAIRE
- OBJET FIXER STYLE BORDURE
- OBJET FIXER STYLE POLICE
- OBJET FIXER TAILLE POLICE
- OBJET FIXER TEXTE EXEMPLE
- OBJET FIXER TITRE
- OBJET FIXER TYPE INDICATEUR
- OBJET FIXER VALEUR MAXIMUM
- OBJET FIXER VALEUR MINIMUM
- OBJET FIXER VISIBLE
- OBJET Lire action
- OBJET Lire activation
- OBJET Lire alignement horizontal
- OBJET Lire alignement vertical
- OBJET LIRE BARRES DEFILEMENT
- OBJET Lire case a cocher trois etats
- OBJET Lire configuration clavier
- OBJET LIRE COORDONNEES
- OBJET Lire correction orthographique
- OBJET LIRE COULEURS RVB
- OBJET LIRE DEFILEMENT
- OBJET LIRE EQUIVALENT CLAVIER
- OBJET LIRE EVENEMENTS
- OBJET Lire feuille de style
- OBJET Lire filtre saisie
- OBJET Lire formatage

OBJET LIRE IMPRESSION TAILLE VARIABLE
OBJET Lire liste reference
OBJET Lire menu contextuel
OBJET Lire message aide
OBJET Lire multiligne
OBJET Lire nom
OBJET Lire nom liste
OBJET LIRE OPTIONS GLISSER DEPOSER
OBJET Lire orientation texte
OBJET Lire pointeur
OBJET Lire police
OBJET Lire rayon arrondi
OBJET Lire rectangle focus invisible
OBJET LIRE REDIMENSIONNEMENT
OBJET Lire saisissable
OBJET Lire source donnees
OBJET LIRE SOUS FORMULAIRE
OBJET Lire style bordure
OBJET Lire style police
OBJET LIRE TAILLE OPTIMALE
OBJET Lire taille police
OBJET LIRE TAILLE SOUS FORMULAIRE
OBJET Lire texte exemple
OBJET Lire titre
OBJET Lire type
OBJET Lire type indicateur
OBJET LIRE VALEUR MAXIMUM
OBJET LIRE VALEUR MINIMUM
OBJET Lire visible
_o_ACTIVER BOUTON
_o_INACTIVER BOUTON

Les commandes de propriétés des objets agissent sur les propriétés des objets présents dans les formulaires. Elles vous permettent de modifier l'apparence et le comportement de ces objets lorsque vous utilisez les formulaires pour afficher les enregistrements et en mode Application.

Important : La portée (l'aire d'action) de ces commandes est le formulaire en cours d'utilisation ; les modifications ne sont pas conservées lorsque vous sortez du formulaire.

Accès aux objets par leur nom d'objet ou à partir du nom de la source de données

Les commandes de propriétés des objets partagent toutes la même syntaxe :

NOM DE LA COMMANDE({*;} objet { ; paramètres spécifiques à la commande)

Si vous spécifiez le paramètre optionnel *, vous indiquez un nom d'objet (une chaîne) dans *objet*.

Vous pouvez utiliser le caractère @ dans ce nom si vous voulez adresser plusieurs objets du formulaire dans un seul appel. Le tableau qui suit montre des exemples de noms d'objets que vous pouvez spécifier pour cette commande.

Noms d'objets	Objets affectés par l'appel
zoneGroupe	Uniquement l'objet zoneGroupe.
zone@	Les objets dont le nom commence par “zone”.
@zoneGroupe	Les objets dont le nom finit par “zoneGroupe”.
@Groupe@	Les objets dont le nom contient “Groupe”.
zone@Btn	Les objets dont le nom commence par “zone” et finit par “Btn”.
@	Tous les objets présents dans le formulaire.

Les noms d'objets de formulaires peuvent contenir jusqu'à 255 octets, permettant la définition et l'application de règles de nommage personnalisées, par exemple "xxxx_Bouton" ou "xxx_Mac".

Note : Il est possible de paramétrer le mode d’interprétation du caractère @, lorsque celui-ci est inclus dans une chaîne de caractères. Cette option influe sur le fonctionnement des commandes du thème "Objets (Formulaires)". Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.

Si vous omettez le paramètre optionnel *, vous indiquez un champ ou une variable dans *objet*. Dans ce cas, vous ne spécifiez pas une chaîne mais une référence de champ ou de variable (champ et variable objet uniquement).

Interaction des commandes génériques avec les textes multistyles

A compter de 4D v14, un nouveau mode d’interaction a été défini entre les commandes génériques telles que **OBJET FIXER COULEURS RVB** ou **OBJET FIXER STYLE POLICE** et les zones de texte multistyle.

Dans les versions précédentes de 4D, l’exécution d’une de ces commandes modifiait le contenu des balises de style personnalisées éventuellement insérées dans la zone. Désormais, seules les propriétés par défaut sont affectées par ces commandes (ainsi que les propriétés stockées via les balises par défaut, le cas échéant). Les balises de style personnalisées sont conservées telles quelles.

Par exemple, soit une zone multistyle dans laquelle les balises par défaut ont été stockées :

Ceci est un mot rouge

Le texte brut de la zone est le suivant :

```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#009900">Ceci est un mot <span
style="color:#D81E05">rouge</span></span>
```

Si vous exécutez le code suivant :

```
OBJET FIXER COULEUR(*;"maZone";-(Bleu+(256*Jaune)))
```

Avec 4D v14, la couleur rouge est préservée :

4D v14

Ceci est un mot rouge

```
<span style="text-align:left;font-
family:'Segoe UI';font-
size:9pt;color:#0000FF">Ceci est un mot
<span style="color:#D81E05">rouge</span>
</span>
```

versions précédentes

Ceci est un mot rouge

```
<span style="font-family:'Segoe UI';font-size:9pt;text-
align:left;font-weight:normal;font-style:normal;text-
decoration:none;color:#0000FF;"><span style="background-
color:#FFFFFF">Ceci est un mot rouge</span></span>
```

Les commandes génériques sont les suivantes :

OBJET FIXER COULEURS RVB
OBJET FIXER COULEUR
OBJET FIXER POLICE
OBJET FIXER STYLE POLICE
OBJET FIXER TAILLE POLICE

Dans le contexte des zones de texte multistyles, les commandes génériques doivent être utilisées pour définir les styles par défaut uniquement. Pour gérer les styles lors de l’exécution de la base, il est recommandé d’utiliser les commandes du thème "**Texte multistyle**".

LIRE INFOS FEUILLE DE STYLE

LIRE INFOS FEUILLE DE STYLE (nomFeuilleStyle ; police ; taille ; styles)

Paramètre	Type		Description
nomFeuilleStyle	Texte	➡	Nom de la feuille de style
police	Texte	⬅	Police de caractères
taille	Entier long	⬅	Taille de police
styles	Entier long	⬅	Valeur de style

Description

La commande **LIRE INFOS FEUILLE DE STYLE** retourne la configuration courante de la feuille de style *nomFeuilleStyle*.

Passez dans *nomFeuilleStyle* le nom de la feuille de style tel que défini en mode Développement. Pour désigner une feuille de style automatique, utilisez une des constantes suivantes, placées dans le thème "**Styles de caractères**" :

Constante	Type	Valeur	Comment
Feuille de style automatique	Chaîne	__automatic__	Utilisée par défaut pour tous les objets
Feuille de style automatique_additionnel	Chaîne	__automatic_additional_text__	Prise en charge par les textes statiques, champs et variables uniquement. Utilisée pour du texte additionnel dans les boîtes de dialogue.
Feuille de style automatique_texte principal	Chaîne	__automatic_main_text__	Prise en charge par les textes statiques, champs et variables uniquement. Utilisée pour le texte principal des boîtes de dialogue.

La commande retourne dans *police* le nom de la police de caractères associée à la feuille de style pour la plate-forme courante.

La commande retourne dans *taille* la taille en points de la police de caractères associée à la feuille de style pour la plate-forme courante.

La commande retourne dans *styles* une valeur correspondant au(x) style(s) associé(s) à la feuille de style pour la plate-forme courante. Vous pouvez comparer la valeur reçue aux constantes suivantes, placées dans le thème "**Styles de caractères**" :

Constante	Type	Valeur
Gras	Entier long	1
Gras et italique	Entier long	3
Gras et souligné	Entier long	5
Italique	Entier long	2
Italique et souligné	Entier long	6
Normal	Entier long	0
Souligné	Entier long	4

Si la commande est exécutée correctement, la variable système *OK* prend la valeur 1. Dans le cas contraire (par exemple si *nomFeuilleStyle* n'existe pas), elle prend la valeur 0.

Exemple

Vous souhaitez connaître la configuration actuelle de la feuille de style "Automatique" :

```
C_ENTIER LONG($taille;$style)
C_TEXTE($pol)
LIRE INFOS FEUILLE DE STYLE(Feuille de style automatique;$pol;$taille;$style)
```

LISTE FEUILLES DE STYLE

LISTE FEUILLES DE STYLE (tabFeuillesStyle)

Paramètre	Type		Description
tabFeuillesStyle	Tableau texte	⇦	Noms des feuilles de style définies dans l'application

Description

La commande **LISTE FEUILLES DE STYLE** retourne la liste des feuilles de style de l'application dans le tableau *tabFeuillesStyle*.

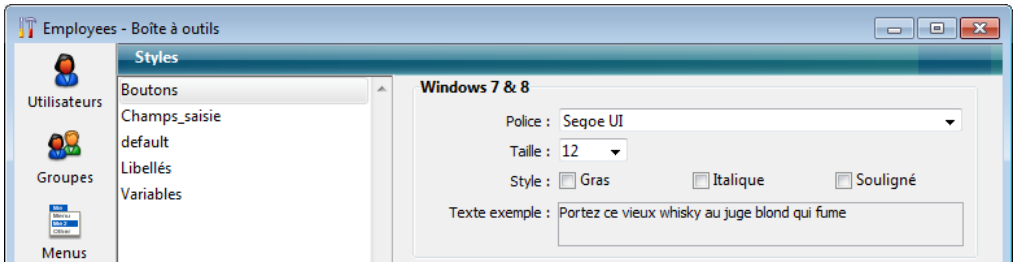
S'il n'a pas été défini auparavant, le tableau texte *tabFeuillesStyle* est créé par la commande. Il est automatiquement dimensionné en fonction du nombre de feuilles de style définies.

A l'issue de l'exécution de la commande, chaque élément du tableau contient un nom de feuille de style. Les noms sont triés par ordre alphabétique, comme dans l'éditeur de feuilles de style. Le premier élément du tableau contient systématiquement "__automatic__", qui représente la feuille de style "Automatique".

Note : Pour des raisons de compatibilité, les feuilles de style automatiques "__automatic_main_text__" et "__automatic_additional_text__" ne sont pas retournées par cette commande. Cependant, elles sont toujours disponibles dans les formulaires.

Exemple

Dans votre application, les feuilles de style suivantes sont définies :



Si vous exécutez le code suivant :

```
LISTE FEUILLES DE STYLE($tTtabstyles)
    //$tTtabstyles{1} contient "__automatic__"
    //$tTtabstyles{2} contient "Boutons"
    //$tTtabstyles{3} contient "Champs_saisie"
    //$tTtabstyles{4} contient "default"
    //$tTtabstyles{5} contient "Libellés"
    //$tTtabstyles{6} contient "Variables"
```

OBJET DEPLACER

OBJET DEPLACER ({* ;} objet ; dépH ; dépV {; redimH {; redimV {; *}}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
dépH	Entier long	⇒	Valeur de déplacement horizontal de l'objet (>0 = vers la droite, <0 = vers la gauche)
dépV	Entier long	⇒	Valeur de déplacement vertical de l'objet (>0 = vers le bas, <0 = vers le haut)
redimH	Entier long	⇒	Valeur de redimensionnement horizontal de l'objet
redimV	Entier long	⇒	Valeur de redimensionnement vertical de l'objet
*	Opérateur	⇒	Si spécifié = coordonnées absolues Si omis = coordonnées relatives

Description

La commande **OBJET DEPLACER** permet de déplacer le ou les objet(s) du formulaire courant, défini(s) par les paramètres * et *objet*, de *dépH* pixels horizontalement et de *dépV* pixels verticalement. Il est également possible (optionnellement) de redimensionner le ou les objet(s) de *redimH* pixels horizontalement et de *redimV* pixels verticalement.

Le sens du déplacement ou du redimensionnement est défini par le signe des valeurs passées dans les paramètres *dépH* et *dépV* :

- Si la valeur est positive, le déplacement ou le redimensionnement s'effectue respectivement vers la droite ou vers le bas.
- Si la valeur est négative, le déplacement ou le redimensionnement s'effectue respectivement vers la gauche ou vers le haut.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne de caractères). Si vous ne passez pas le paramètre *, vous indiquez que *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable de type objet uniquement).
Si vous passez un nom d'objet dans le paramètre *objet* et utilisez le caractère joker @ afin de sélectionner plusieurs objets, tous les objets sélectionnés seront déplacés ou redimensionnés.

Par défaut, les valeurs de *dépH*, *dépV*, *redimH* et *redimV* modifient les coordonnées de l'objet relativement à sa position précédente. Si vous souhaitez que ces paramètres définissent des coordonnées absolues, passez le dernier paramètre optionnel *.

Cette commande fonctionne uniquement dans les contextes suivants :

- Formulaires entrée en mode saisie,
- Formulaires affichés via la commande **DIALOGUE**,
- En-têtes et pieds de page des formulaires sortie affichés par les commandes **MODIFIER SELECTION** ou **VISUALISER SELECTION**,
- Formulaires en cours d'impression.

Exemple 1

L'instruction suivante déplace le bouton "Bouton_1" de 10 pixels vers la droite et de 20 pixels vers le haut, et agrandit le bouton de 30 pixels en largeur et de 40 en hauteur :

```
OBJET DEPLACER(*;"Bouton_1";10;-20;30;40)
```

Exemple 2

L'instruction suivante place le bouton "Bouton_1" aux coordonnées (10;20) (30;40) :

```
OBJET DEPLACER(*;"Bouton_1";10;20;30;40;*)
```

OBJET DUPLIQUER

OBJET DUPLIQUER ({* ;} objet {; nouvNom {; nouvVar {; reliéA {; dépH {; dépV {; redimH {; redimV}}}}}}} {; *})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
nouvNom	Texte	⇒	Nom du nouvel objet
nouvVar	Pointeur	⇒	Pointeur vers la variable du nouvel objet
reliéA	Texte	⇒	Nom de l'objet saisissable (ou du bouton radio) précédent
dépH	Entier long	⇒	Décalage horizontal du nouvel objet (>0 = vers la droite, <0 = vers la gauche)
dépV	Entier long	⇒	Décalage vertical du nouvel objet (>0 = vers le bas, <0 = vers le haut)
redimH	Entier long	⇒	Valeur de redimensionnement horizontal du nouvel objet
redimV	Entier long	⇒	Valeur de redimensionnement vertical du nouvel objet
*	Opérateur	⇒	Si spécifié = coordonnées absolues Si omis = coordonnées relatives

Description

La commande **OBJET DUPLIQUER** permet de créer une copie de l'objet désigné par le paramètre *objet* dans le contexte du formulaire en cours d'exécution (mode Application). Le formulaire d'origine, généré en mode Développement, n'est pas modifié.

Par défaut, toutes les options définies dans la Liste des propriétés pour l'objet source sont appliquées à la copie (taille, options de redimensionnement, couleur, etc.), y compris la méthode objet éventuellement associée. Les exceptions suivantes sont toutefois à noter :

- Bouton par défaut : il ne peut y avoir qu'un seul bouton par défaut dans un formulaire. Lorsque vous dupliquez un bouton ayant la propriété "Bouton par défaut", cette propriété est attribuée à la copie et est supprimée de l'objet d'origine.
- Equivalents clavier : le raccourci clavier associé à un objet source n'est pas dupliqué. Cette propriété est laissée vide dans la copie.
- Noms d'objet : il ne peut pas y avoir plusieurs objets de même nom dans un formulaire. Si vous ne passez pas le paramètre *nouvNom*, le nom de l'objet source est automatiquement incrémenté dans le nouvel objet (cf. ci-dessous).

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre *objet* désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables objets uniquement).
Si vous passez une référence de champ ou de variable et si le formulaire contient plusieurs objets utilisant la même référence, la première occurrence trouvée est utilisée. Dans ce cas, pour éviter toute ambiguïté, il est conseillé de travailler avec les noms d'objets, qui sont uniques.

Passez dans le paramètre *nouvNom* le nom attribué à la copie de l'objet. Ce nom doit être conforme aux règles de nommage des objets et être unique dans le formulaire. S'il est invalide ou est déjà utilisé par un autre objet, la commande ne fait rien et la variable *OK* retourne 0.
Si vous omettez ce paramètre ou passez une chaîne vide, le nouveau nom est automatiquement généré par incrémentation du nom de l'objet source (si ce nom n'est pas déjà utilisé). Par exemple :

Nom d'origine	Nom de la copie
Bouton	Bouton 1
Bouton20	Bouton21
Bouton21	Bouton23 si Bouton22 existe déjà

Passez dans *nouvVar* un pointeur vers la variable à associer au nouvel objet. Vous devez en principe pointer vers une variable du même type que celle de l'objet d'origine mais certains "retypages" sont possibles. La commande propose des automatismes facilitant l'écriture de code générique :

- De manière générale, toutes les variables "saisissables" peuvent être retypées ; par exemple, un objet affichant une Date ou un Entier long peut être dupliqué et utilisé avec une variable de type Texte. Les propriétés compatibles sont maintenues. La commande permet également les changements de type entre des objets Texte et des objets Image. A noter qu'un objet texte dupliqué et associé à une variable ou un champ booléen aura automatiquement l'apparence d'une case à cocher.
- Il est généralement possible de transformer dynamiquement une variable en champ et inversement.
  En revanche, les objets graphiques (boutons, cases à cocher...) ne peuvent pas être tranformés en d'autres types de contrôles.

Si le type de la variable est incompatible avec l'objet, la commande ne fait rien et la variable *OK* prend la valeur 0. Si vous omettez ce paramètre, la variable est créée dynamiquement par 4D (cf. paragraphe "Variables dynamiques" dans la section **Variables**). Si vous dupliquez un objet statique (ligne, rectangle, image statique...) ce paramètre est ignoré. Passez un pointeur Nil (->[]) si vous souhaitez pouvoir utiliser les autres paramètres.

Vous utilisez le paramètre *reliéA* dans deux cas :

- mise à jour de l'ordre de saisie : dans ce cas, passez dans *reliéA* le nom de l'objet saisissable situé juste avant l'objet dupliqué. Si vous souhaitez que le nouvel objet devienne le premier objet dans l'ordre de saisie de la page, passez la constante Objet Premier ordre saisie (cf. commande **OBJET Lire pointeur**).
- association à un groupe de boutons radio : les boutons radios fonctionnent de façon coordonnée lorsqu'ils sont groupés. Si l'objet dupliqué est un bouton radio, passez dans *reliéA* le nom d'un bouton radio du groupe auquel rattacher le nouvel objet.

Si vous omettez ce paramètre ou passez une chaîne vide, le nouvel objet devient le dernier objet saisissable de la page du formulaire. Dans le cas d'un bouton radio, l'objet est rattaché au groupe du bouton source.

Le nouvel objet peut être déplacé et redimensionné via les paramètres *dépH*, *dépV*, *redimH* et *redimV*. Comme pour la commande **OBJET DEPLACER**, le sens du déplacement ou du redimensionnement est défini par le signe des valeurs passées dans les paramètres *dépH* et *dépV* :

- Si la valeur est positive, le déplacement ou le redimensionnement s'effectue respectivement vers la droite ou vers le bas.
- Si la valeur est négative, le déplacement ou le redimensionnement s'effectue respectivement vers la gauche ou vers le haut.

Par défaut, les valeurs de *dépH*, *dépV*, *redimH* et *redimV* modifient les coordonnées de l'objet relativement à sa position précédente. Si vous souhaitez que ces paramètres définissent des coordonnées absolues, passez le dernier paramètre optionnel *.
Si vous omettez ces paramètres, le nouvel objet se superpose à l'objet d'origine.

Cette commande doit être utilisée dans le contexte de l'affichage d'un formulaire. Elle sera généralement appelée dans l'événement Sur chargement du formulaire ou suite à une action utilisateur (événement Sur clic).

Note : Si l'événement Sur chargement est associé à l'objet d'origine, il est généré pour l'objet dupliqué au moment de l'exécution de la commande. Ce principe permet par exemple d'initialiser la valeur de l'objet.

Pour des raisons techniques et logiques, **OBJET DUPLIQUER** ne peut pas être appelée dans le cadre de certains événements formulaire, notamment :

- Evénement Sur chargement généré dans une méthode objet
- Evénement Sur libération.
- Evénement lié à un contexte d'impression (Sur entête, Sur impression corps, etc.). Pour imprimer plusieurs fois un objet, vous devez utiliser la commande **Imprimer objet**.

Lorsque la commande est appelée dans un contexte non pris en charge, l'objet n'est pas dupliqué et la variable OK prend la valeur 0. Si elle est appelée dans un contexte d'impression, l'erreur -10601 est en outre générée.

Si la commande est exécutée correctement, la variable *OK* prend la valeur 1. Sinon, elle prend la valeur 0.

Exemple 1

Création d'un nouveau bouton nommé "BoutonAnnul" au-dessus de l'objet existant "BoutonOK" et association à la variable *vAnnul* :

```
OBJET DUPLIQUER(*;"BoutonOK";"BoutonAnnul";vAnnul)
```

Exemple 2

Création d'un nouveau bouton radio "bRadio6" basé sur le bouton radio existant "bRadio5". Ce bouton sera associé à la variable <>r6, intégré au groupe du bouton "bRadio5" et placé 20 pixels au-dessous :

```
OBJET DUPLIQUER(*;"bRadio5";"bRadio6";<>r6;"bRadio5";0;20)
```

OBJET Est un texte style

OBJET Est un texte style ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Booléen	⮌	Vrai si l'objet est un texte en multistyle, Faux sinon

Description

La commande **OBJET Est un texte style** retourne **Vrai** si l'option "Multistyle" est cochée pour l'objet ou les objets désigné(s) par les paramètres *objet* et *.

L'option "Multistyle" permet d'utiliser des zones de texte riche (rich text) comportant des variations de style individuelles. Pour plus d'informations, reportez-vous à la section **LISTE SOURCES DONNEES** dans le manuel *Mode Développement*.

Les objets multistyles peuvent être gérés par programmation à l'aide des commandes du thème "**Texte multistyle**".

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Note : La commande **OBJET Est un texte style** retourne **Vrai** lorsqu'elle est appliquée à une zone 4D Write Pro.

Exemple

Un formulaire comporte un champ représenté par deux objets différents, l'un avec la propriété "Multistyle" cochée, l'autre sans cette propriété cochée. Vous pouvez écrire :

```
$Style:=OBJET Est un texte style(*;"Texte_stylé")
    // retourne Vrai (l'option "Multistyle" est cochée)

$Style:=OBJET Est un texte style(*;"Texte_brut")
    // retourne Faux (l'option "Multistyle" n'est cochée)
```

OBJET FIXER ACTION

OBJET FIXER ACTION ({* ;} objet ; action)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
action	Texte	⇒	Action à associer

Description

La commande **OBJET FIXER ACTION** vous permet de modifier, pour le process courant, l'action standard associée à l'objet ou aux objets désigné(s) par les paramètres *objet* et *.

Note : Les actions standard peuvent également être définies via la Liste des propriétés en mode Développement.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans le paramètre *action* un code d'action (une chaîne) désignant l'action standard à associer à l'objet. Vous pouvez passer une des constantes suivantes, placées dans le thème "**Valeurs Texte pour Action standard associée**" :

Constante	Type	Valeur
Objet action Actualiser URL courant	Chaîne	39
Objet action Afficher Presse papiers	Chaîne	23
Objet action Ajouter sous enreg	Chaîne	14
Objet action Aller à page	Chaîne	15
Objet action Annuler	Chaîne	1
Objet action Annuler édition	Chaîne	17
Objet action Arrêter chargement URL	Chaîne	40
Objet action Coller	Chaîne	20
Objet action Copier	Chaîne	19
Objet action Couper	Chaîne	18
Objet action CSM	Chaîne	36
Objet action Dernier enreg	Chaîne	6
Objet action Dernière page	Chaîne	11
Objet action Effacer	Chaîne	21
Objet action Enreg précédent	Chaîne	4
Objet action Enreg suivant	Chaîne	3
Objet action Modifier sous enreg	Chaîne	12
Objet action Ouvrir URL précédent	Chaîne	37
Objet action Ouvrir URL suivant	Chaîne	38
Objet action Page précédente	Chaîne	9
Objet action Page suivante	Chaîne	8
Objet action Premier enreg	Chaîne	5
Objet action Première page	Chaîne	10
Objet action Propriétés de la base	Chaîne	32
Objet action Quitter	Chaîne	27
Objet action Répéter	Chaîne	31
Objet action Retour au mode Développement	Chaîne	35
Objet action Séparateur auto	Chaîne	16
Objet action Supprimer enreg	Chaîne	7
Objet action Supprimer sous enreg	Chaîne	13
Objet action Test application	Chaîne	26
Objet action Tout sélectionner	Chaîne	22
Objet action Valider	Chaîne	2
Objet sans action standard	Chaîne	0

Exemple

Vous souhaitez associer l'action standard **Valider** à un bouton :

```
OBJET FIXER ACTION(*;"bValidate";Objet action Valider)
```

OBJET FIXER ACTIVATION

OBJET FIXER ACTIVATION ({* ;} objet ; actif)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
actif	Booléen	⇒	Vrai = objet(s) activé(s), Faux sinon

Description

La commande **OBJET FIXER ACTIVATION** permet d'activer ou d'inactiver l'objet ou le groupe d'objets désigné par *objet* dans le formulaire courant.

Un objet activé réagit aux clics souris et aux raccourcis clavier.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable (variable objet uniquement).

Passez Vrai dans le paramètre *actif* pour activer les objets et Faux pour les inactiver.

Cette commande peut être appliquée aux types d'objets suivants :

- Bouton, Bouton par défaut, Bouton 3D, Bouton invisible, Bouton inversé
- Bouton radio, Bouton radio 3D, Bouton image
- Case à cocher, Case à cocher 3D
- Pop-up menu, Liste déroulante, Combo Box, Menu/Liste déroulante
- Thermomètre, Règle

Note : Cette commande est sans effet avec un objet auquel une action standard a été assignée (4D se charge de modifier l'état de cet objet lorsque c'est nécessaire), sauf dans le cas des actions **Valider** et **Annuler**.

OBJET FIXER ALIGNEMENT HORIZONTAL

OBJET FIXER ALIGNEMENT HORIZONTAL ({* ;} objet ; alignement)

Paramètre	Type		Description
*	Opérateur	⇨	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇨	Nom d'objet (si * est passé) ou Champ ou variable (si * est omis)
alignement	Entier long	⇨	Code d'alignement

Description

La commande **OBJET FIXER ALIGNEMENT HORIZONTAL** vous permet de fixer le type d'alignement horizontal appliqué à l'objet ou aux objets désigné(s) par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre *objet* désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables de formulaire uniquement).

Passez dans le paramètre *alignement* une des constantes du thème **Objets de formulaire (Propriétés)** :

Constante	Type	Valeur	Comment
Aligné à droite	Entier long	4	
Aligné à gauche	Entier long	2	
Aligné au centre	Entier long	3	
Aligné par défaut	Entier long	1	
wk justify	Entier long	5	Alignement justifié. Disponible uniquement pour les zones 4D Write Pro.

Note : La constante wk justify est disponible dans le thème "**4D Write Pro**".

Les objets de formulaire auxquels vous pouvez appliquer cette commande sont les suivants :

- Zones de défilement
- Combo box
- Textes statiques
- Zones de groupes
- Pop up menus/Listes déroulantes
- Champs
- Variables
- List box
- Colonnes de list box
- En-tête de list box
- Pieds de list box
- Zones **4D Write Pro**

OBJET FIXER ALIGNEMENT VERTICAL

OBJET FIXER ALIGNEMENT VERTICAL ({* ;} objet ; alignement)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
alignement	Entier long	⇒	Code d'alignement

Description

La commande **OBJET FIXER ALIGNEMENT VERTICAL** vous permet de modifier par programmation le type d'alignement vertical appliqué à l'objet ou aux objets désigné(s) par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Passez dans le paramètre *alignement* une des constantes suivantes du thème **Objets de formulaire (Propriétés)** :

Constante	Type	Valeur	Comment
Aligné au centre	Entier long	3	
Aligné en bas	Entier long	4	
Aligné en haut	Entier long	2	
Aligné par défaut	Entier long	1	

Les objets de formulaire auxquels un alignement vertical peut être appliqué sont les suivants :

- list box,
- colonnes de list box,
- en-tête et pieds de list box.

OBJET FIXER BARRES DEFILEMENT

OBJET FIXER BARRES DEFILEMENT ({* ;} objet ; horizontal ; vertical)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
horizontal	Booléen, Entier long	⇒	Visibilité de la barre horizontale
vertical	Booléen, Entier long	⇒	Visibilité de la barre verticale

Description

La commande **OBJET FIXER BARRES DEFILEMENT** permet d'afficher ou de masquer les barres de défilement horizontale et/ou verticale dans l'objet désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Propriétés des objets**.

Passez dans les paramètres *horizontal* et *vertical* des valeurs indiquant si les barres de défilement correspondantes doivent être affichées. Vous pouvez passer soit des valeurs booléennes (Vrai=affichée, Faux=masquée), soit des valeurs numériques (0=masquée, 1=affichée, 2=mode automatique). Utiliser des valeurs numériques permet notamment d'accéder au mode automatique, dans lequel la barre de défilement n'apparaît que lorsque c'est nécessaire.

Le tableau suivant indique les valeurs que vous pouvez passer dans les paramètres *horizontal* et *vertical* pour les objets acceptant des barres de défilement (le mode automatique n'est pas disponible avec tous les objets) :

Objets avec barres de défilement	Masquer barre	Afficher barre	Mode automatique
Champs et variables objet texte	Faux ou 0	Vrai ou 1	*non disponible*
Champs et variables objet image	Faux ou 0	Vrai ou 1	2
List box	Faux ou 0	Vrai ou 1	2
Listes hiérarchiques	Faux ou 0	Vrai ou 1	2
Sous-formulaires	Faux ou 0	Vrai ou 1	*non disponible*

Par défaut, les barres de défilement sont affichées.

Note : Pour plus d'informations sur le mode automatique, reportez-vous à la section **Barres de défilement**.

OBJET FIXER CASE A COCHER TROIS ETATS

OBJET FIXER CASE A COCHER TROIS ETATS ({* ;} objet ; troisEtats)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
troisEtats	Booléen	⇒	Vrai = case à cocher à trois états, Faux = case à cocher standard

Description

La commande **OBJET FIXER CASE A COCHER TROIS ETATS** vous permet de modifier, pour le process courant, la propriété "Trois états" de la ou des case(s) à cocher désignée(s) par les paramètres *objet* et *.

L'option "Trois états" permet d'utiliser l'état supplémentaire "semi-coché" pour les cases à cocher. Pour plus d'informations, reportez-vous au paragraphe **Cases à cocher à trois états** dans le manuel *Mode Développement*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Cette commande s'applique uniquement aux case à cocher associées à des variables, et non aux champs booléens représentés sous forme de cases à cocher.

Passez **Vrai** dans le paramètre *troisEtat* pour activer le mode "trois états", ou **Faux** pour le désactiver.

OBJET FIXER CONFIGURATION CLAVIER

OBJET FIXER CONFIGURATION CLAVIER ({* ;} objet ; codeLangue)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
codeLangue	Chaîne	⇒	Code de langue RFC3066 ISO639 et ISO3166, "" = pas de changement

Description

La commande **OBJET FIXER CONFIGURATION CLAVIER** vous permet de définir ou de modifier dynamiquement la configuration clavier associée à l'objet ou aux objets désigné(s) par les paramètres *objet* et * pour le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable ou un champ. Dans ce cas, vous ne passez pas une nom mais une référence.

Passez dans le paramètre *codeLangue* une chaîne indiquant le code de langue à utiliser, basé sur les RFC3066, ISO639 et ISO3166. Pour plus d'informations, reportez-vous à la description de la commande **FIXER LANGUE BASE**.

OBJET FIXER COORDONNEES

OBJET FIXER COORDONNEES ({* ;} objet ; gauche ; haut {; droite ; bas})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Entier long	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
gauche	Entier long	⇒	Coordonnée gauche de l'objet en pixels
haut	Entier long	⇒	Coordonnée supérieure de l'objet en pixels
droite	Entier long	⇒	Coordonnée droite de l'objet en pixels
bas	Entier long	⇒	Coordonnée inférieure de l'objet en pixels

Description

La commande **OBJET FIXER COORDONNEES** permet de modifier l'emplacement et, optionnellement, la taille de l'objet ou des objets désigné(s) par les paramètres *objet* et * pour le process courant.

Note : Cette commande équivaut à utiliser la commande **OBJET DEPLACER** en passant le 2e paramètre *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans les paramètres *gauche* et *haut* les nouvelles coordonnées absolues de l'objet dans le formulaire. Ces coordonnées doivent être exprimées en pixels par rapport à l'angle supérieur gauche du formulaire.

Vous pouvez également passer des valeurs de coordonnées absolues dans les paramètres *droite* et *bas*, indiquant l'angle inférieur droit de l'objet. Si cet angle ne correspond pas à celui de l'objet après application des paramètres *gauche* et *haut*, l'objet est redimensionné en conséquence.

Note : Si vous souhaitez déplacer un objet relativement à sa position initiale, il est préférable d'utiliser la commande existante **OBJET DEPLACER**.

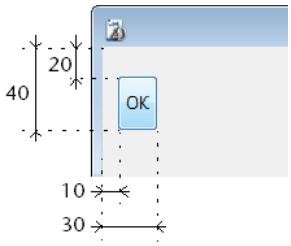
Cette commande fonctionne uniquement dans les contextes suivants :

- Formulaires entrée en mode saisie,
- Formulaires affichés via la commande **DIALOGUE**,
- En-têtes et pieds de page des formulaires sortie affichés par la commande **MODIFIER SELECTION** ou **VISUALISER SELECTION**,
- Formulaires en cours d'impression.

Exemple

L'instruction suivante place l'objet "bouton_1" aux coordonnées (10,20) (30,40) :

```
OBJET FIXER COORDONNEES(*;"bouton_1";10;20;30;40)
```



OBJET FIXER CORRECTION ORTHOGRAPHIQUE

OBJET FIXER CORRECTION ORTHOGRAPHIQUE ({* ;} objet ; correctionAuto)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
correctionAuto	Booléen	⇒	Vrai = correction automatique, Faux = pas de correction automatique

Description

La commande **OBJET FIXER CORRECTION ORTHOGRAPHIQUE** permet de définir ou de modifier dynamiquement le statut de l'option **Correction orthographique** du ou des objet(s) désigné(s) par les paramètres *objet* et * pour le process courant. Cette option permet d'activer ou non la correction orthographique automatique lors de la saisie pour l'objet (objets de type texte uniquement).

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable ou un champ. Dans ce cas, vous ne passez pas un nom mais une référence.

Passez **Vrai** dans *correctionAuto* pour activer la correction automatique pour *objet*, et **Faux** pour la désactiver.

OBJET FIXER COULEUR

OBJET FIXER COULEUR ({* ;} objet ; couleur {; couleurAlt})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Champ, Variable	⇒	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
couleur	Entier long	⇒	Nouvelles couleurs pour l'objet
couleurAlt	Entier long	⇒	Couleur de fond alternée pour une list box

Description

La commande **OBJET FIXER COULEUR** définit les couleurs de premier plan et d'arrière-plan du ou des objet(s) de formulaire spécifié(s) par *objet*. Si *objet* est une list box, un paramètre supplémentaire permet de définir les couleurs de premier plan et d'arrière-plan des lignes paires (couleurs alternées).

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Le paramètre *couleur* définit à la fois les couleurs de premier plan et d'arrière-plan. La couleur est calculée de la manière suivante : **Couleur:=-(Premier_Plan+(256 * Arrière_Plan))**, où **Premier_Plan** et **Arrière_Plan** sont des numéros de couleur (de 0 à 255) parmi la palette de couleurs de 4D — que vous pouvez visualiser, par exemple, dans la Liste des propriétés de l'éditeur de formulaires. **Couleur** est toujours un nombre négatif.
Par exemple, si la couleur de premier plan est 20 et si la couleur d'arrière-plan est 10, alors *couleur* est égal à - (20 + (256 * 10)) soit -2580.

couleurAlt permet de désigner une couleur de fond alternative pour les lignes paires d'une list box ou d'une colonne de list box. Vous devez passer dans *couleurAlt* uniquement la partie "arrière-plan" de la formule de couleur, c'est-à-dire **CouleurAlt:=-(256 * Arrière_Plan)**.
Lorsque ce paramètre est passé, la partie "arrière-plan" du paramètre *couleur* s'applique aux lignes impaires uniquement. Utiliser des couleurs de fond alternées améliore la lisibilité des tableaux. Si *objet* désigne l'objet list box, les couleurs alternées sont utilisées dans la totalité de la list box. Si *objet* désigne une colonne, seule la colonne utilisera les couleurs définies.

Les numéros les plus souvent utilisés sont fournis par 4D sous forme de constantes prédéfinies, placées dans le thème "**Couleurs**" :

Constante	Type	Valeur
Blanc	Entier long	0
Bleu	Entier long	6
Bleu clair	Entier long	7
Bleu foncé	Entier long	5
Gris	Entier long	14
Gris clair	Entier long	12
Gris foncé	Entier long	11
Jaune	Entier long	1
Marron	Entier long	13
Marron foncé	Entier long	10
Noir	Entier long	15
Orange	Entier long	2
Rouge	Entier long	3
Vert	Entier long	8
Vert foncé	Entier long	9
Violet	Entier long	4

Note : Tandis que **OBJET FIXER COULEUR** travaille avec des couleurs indexées dans la palette de couleurs de 4D, la commande **OBJET FIXER COULEURS RVB** vous permet de travailler avec toute couleur RVB. Pour rétablir les couleurs automatiques d'un objet, utilisez la commande **OBJET FIXER COULEURS RVB** avec les constantes Coul premier plan et Coul arrière plan.

Exemple 1

L'exemple suivant définit la couleur de la zone de texte représentée ci-dessous dans l'éditeur de formulaires :

Après l'exécution de cette instruction :

```
OBJET FIXER COULEUR(*;"Montexte";-(Jaune+(256*Rouge)))
```

... la zone prend l'apparence suivante :

Exemple 2

Vous voulez définir une couleur de fond alternée pour une colonne de list box. Vous pouvez écrire :

```
OBJET FIXER COULEUR(*;"countryCol";-(Bleu foncé+(256*Rouge));-(256*Orange))
```


Country
Angola
Argentina
Australia
Brazil
Canada
Chile
China
Egypt
France
Germany
India

OBJET FIXER COULEURS RVB

OBJET FIXER COULEURS RVB ({* ;} objet ; couleurAvantPlan ; couleurArrièrePlan {; couleurArrièrePlanAlt})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou Variable (si * est omis)
couleurAvantPlan	Entier long	⇒	Valeur de la couleur RVB d'avant-plan
couleurArrièrePlan	Entier long	⇒	Valeur de la couleur RVB d'arrière-plan
couleurArrièrePlanAlt	Entier long	⇒	Valeur de la couleur RVB d'arrière-plan alternée

Description

La commande **OBJET FIXER COULEURS RVB** modifie les couleurs d'avant-plan et d'arrière-plan du ou des objet(s) défini(s) par le paramètre *objet* et le paramètre optionnel *. Lorsque la commande est appliquée à un objet de type List box, un paramètre supplémentaire permet de modifier la couleur alternée des lignes.

Si vous passez le paramètre optionnel *, vous spécifiez que le paramètre *objet* est le nom d'un objet (une chaîne de caractères). Si le paramètre * est omis, vous spécifiez que *objet* est un champ ou un objet. Dans ce cas, vous ne passez pas dans *objet* une chaîne de caractères mais la référence à un champ ou à une variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Le paramètre facultatif *couleurArrièrePlanAlt* permet de désigner une couleur alternative pour l'arrière-plan (c'est-à-dire le fond) des lignes paires. Ce paramètre n'est utile que lorsque l'objet désigné est de type List box ou colonne de list box. Lorsque ce paramètre est utilisé, la *couleurArrièrePlan* est utilisée pour le fond des lignes impaires uniquement. Utiliser des couleurs alternées améliore la lisibilité des tableaux.
Si *objet* désigne l'objet List box, les couleurs alternées sont utilisées dans la totalité de la list box. Si *objet* désigne une colonne de list box, seule la colonne utilisera les couleurs définies.

Vous passez des valeurs de couleurs RVB dans les paramètres *couleurAvantPlan*, *couleurArrièrePlan* et *couleurArrièrePlanAlt*. Ces valeurs sont des entiers longs de 4 octets dont le format (*0x00RRGGBB*) est décrit ci-dessous (les octets sont numérotés de 0 à 3 de la droite vers la gauche) :

Octet	Description
3	Doit être zéro pour une couleur RVB absolue
2	Composante rouge de la couleur (0..255)
1	Composante verte de la couleur (0..255)
0	Composante bleue de la couleur (0..255)

Le tableau ci-dessous présente des exemples de valeurs de couleurs RVB :

Valeur	Description
0x00000000	Noir
0x00FF0000	Rouge vif
0x0000FF00	Vert vif
0x000000FF	Bleu vif
0x007F7F7F	Gris
0x00FFFF00	Jaune vif
0x00FF7F7F	Rouge pastel
0x00FFFFFF	Blanc

Vous pouvez aussi spécifier une des couleurs "système" utilisées par défaut par 4D pour dessiner des objets ayant la propriété de couleur "automatique". Les constantes prédéfinies suivantes sont proposées par 4D dans le thème "**FIXER COULEUR RVB**" :

Constante	Type	Valeur	Comment
Coul arrière plan	Entier long	-2	
Coul claire	Entier long	-4	
Coul de fond texte sélect	Entier long	-7	
Coul fond élément sélect désact	Entier long	-11	
Coul fond ligne menu sélect	Entier long	-9	
Coul fond transparent	Entier long	-16	Cette constante peut être utilisée uniquement avec les paramètres *couleurArrièrePlan* et *couleurArrièrePlanAlt*.
Coul premier plan	Entier long	-1	
Coul sombre	Entier long	-3	
Coul texte ligne menu sélect	Entier long	-10	
Coul texte sélect	Entier long	-8	

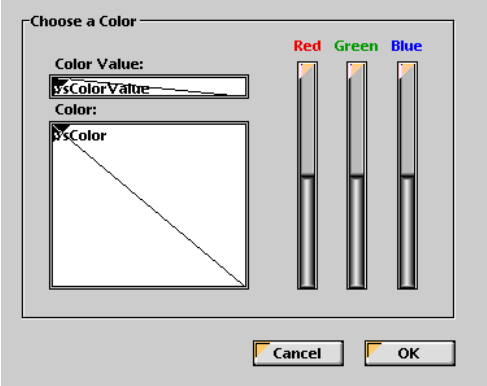
Par exemple, vous pouvez obtenir les couleurs suivantes pour les objets de type champ ou variable saisissable sur des systèmes standard :

Windows		MacOS
	Coul premier plan	
	Coul arrière plan	
	Coul sombre	
	Coul claire	
	Coul de fond texte sélect	
	Coul texte sélect	
	Coul fond ligne menu sélect	
	Coul texte ligne menu sélect	
	Coul fond élément sélect désact	

ATTENTION : Ces couleurs automatiques dépendent du système et du type d'objet auquel elles sont affectées. En fonction de la version de l'OS ou si vous personnalisez vos couleurs système, les couleurs automatiques de 4D seront modifiées en conséquence. Utilisez les valeurs de couleurs automatiques pour assigner à des objets les couleurs système, et non pour leur assigner les mêmes couleurs que celles de l'exemple ci-dessus.

Exemple 1

Voici un formulaire contenant deux variables non saisissables, *vsColorValue* et *vsColor* ainsi que trois thermomètres, *thRouge*, *thVert* et *thBleu* :



Les méthodes associées à ces objets sont les suivantes :

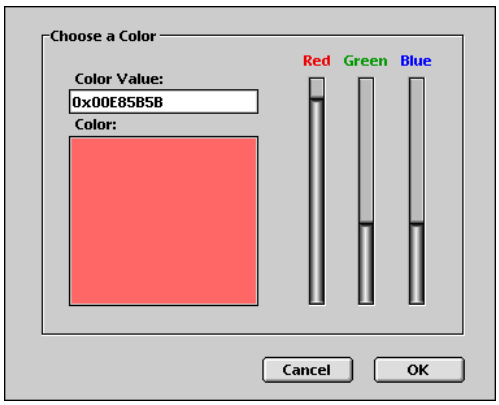
```
  ` Méthode objet de la variable non saisissable vsColorValue
Au cas ou
   :(Evenement formulaire=Sur chargement)
      vsColorValue:="0x00000000"
Fin de cas

  ` Méthode objet de la variable non saisissable vsColor
Au cas ou
   :(Evenement formulaire=Sur chargement)
      vsColor:=""
      OBJET FIXER COULEURS RVB(vsColor;0x00FFFFFF;0x0000)
Fin de cas

  ` Méthode objet du thermomètre thRouge
CLIC SUR THERMOMETRE COULEUR

  ` Méthode objet du thermomètre thVert
CLIC SUR THERMOMETRE COULEUR

  ` Méthode objet du thermomètre thBleu
CLIC SUR THERMOMETRE COULEUR
```

La méthode projet appelée par les trois thermomètres est la suivante :

```
  ` Méthode projet CLIC SUR THERMOMETRE COULEUR
OBJET FIXER COULEURS RVB(vsColor;0x00FFFFFF;(thRouge << 16)+(thVert << 8)+thBleu)
vsColorValue:=Chaine((thRouge << 16)+(thVert << 8)+thBleu;"&x")
Si(thRouge=0)
   vsColorValue:=Sous chaine(vsColorValue;1;2)+"0000"+Sous chaine(vsColorValue;3)
Fin de si
```

Notez l'utilisation des **Opérateurs sur les bits** pour le calcul des valeurs des couleurs à partir de celles des thermomètres.

En exécution, le formulaire a l'aspect suivant :



Exemple 2

Passage du fond en transparent avec couleur de police claire :


```
OBJET FIXER COULEURS RVB(*;"maVar";Coul claire;Coul fond transparent)
```

OBJET FIXER DEFILEMENT

OBJET FIXER DEFILEMENT (* ; objet {; positionLigne {; positionH}}{; *})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une table, un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Table, champ ou variable (si * est omis)
positionLigne	Entier long	⇒	Numéro de ligne à afficher ou Défilement vertical en pixels (images)
positionH	Entier long	⇒	Numéro de colonne à afficher (list box) ou Défilement horizontal en pixels (images)
*	Opérateur	⇒	Afficher la ligne (et la colonne si le paramètre positionH est passé) en première position après défilement (listes) Appliquer un défilement relatif (images)

Description

La commande **OBJET FIXER DEFILEMENT** permet de faire défiler le contenu de plusieurs types d'objets : lignes d'un sous-formulaire, d'un formulaire liste affiché via la commande **MODIFIER SELECTION** ou **VISUALISER SELECTION**, ou d'une liste hiérarchique, lignes et colonnes d'une List box ou encore pixels d'une image.

Note : Le défilement par programmation d'un objet reste possible même si les barres de défilement ont été masquées dans le formulaire.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *objet* est le nom d'un objet de type sous-formulaire, liste hiérarchique, List box ou champ/variable image (dans ce cas, passez une chaîne dans *objet*). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une table (table du formulaire liste ou du sous-formulaire), une variable (*RefListe* de liste hiérarchique, list box ou image) ou un champ.

Le paramètre *positionLigne* permet de spécifier le numéro de la ligne à afficher ou, dans le cas d'une image, la coordonnée verticale du pixel à afficher.
Si vous ne passez pas ce paramètre, la commande provoque le défilement vertical des lignes de la liste de manière à ce que la première ligne sélectionnée (surlignée) dans la liste soit visible. Dans ce cas, si aucune ligne n'est sélectionnée ou si au moins une ligne sélectionnée est déjà visible, aucun défilement vertical n'est effectué.
Si vous passez ce paramètre, la commande provoque le défilement vertical des lignes de la liste de manière à ce que la ligne désignée soit visible (qu'elle soit surlignée ou non). Si la ligne est déjà visible, la commande ne fait rien, sauf si le second paramètre * est passé (cf. ci-dessous).

- Pour les formulaires liste et les sous-formulaires, ce numéro correspond au numéro d'un enregistrement parmi la sélection courante, c'est-à-dire sa position.
- Dans le cas des listes hiérarchiques, la commande tient compte de l'état déployé/contracté des éléments.
- Pour les list box, ce numéro correspond au numéro de la ligne parmi toutes les lignes de l'objet (y compris les lignes éventuellement cachées). Si le numéro passé dans *positionLigne* correspond à une ligne masquée dans la list box, la commande affiche la première ligne visible suivante.

  Note : Gardez à l'esprit que cette commande se base toujours sur la représentation "standard" (non hiérarchique) d'une list box, même si elle est affichée en mode hiérarchique. Par conséquent, le résultat pourra être différent suivant que la list box est affichée en mode standard ou en mode hiérarchique (cf. exemple).
- Dans le cas d'une image affichée dans le formulaire, *positionLigne* indique le point de coordonnée verticale de l'image à afficher dans l'objet. Passez 0 dans *positionLigne* dans pour ne pas faire défiler l'image dans la dimension verticale. La valeur doit être exprimée en pixels relativement à l'origine de l'image. Si le point de coordonnée verticale est déjà visible dans l'objet, la commande ne fait rien (hormis si vous passez le second paramètre *, cf. ci-dessous). L'image doit être affichée dans le format "Image tronquée (non centrée)".

Le paramètre *positionH* peut être utilisé dans le contexte d'une list box ou d'une image.

- Dans le cas d'une list box, vous pouvez passer dans *positionH* un numéro de colonne. L'exécution de la commande provoquera le défilement horizontal de la list box de manière à ce que la colonne soit visible. Si la colonne est déjà visible, la commande ne fait rien. Comme pour le défilement vertical, si vous passez le second paramètre optionnel *, la colonne rendue visible par la commande (si la list box a effectivement défilé) sera placée en première position (cf. ci-dessous).
- Dans le cas d'une image affichée dans le formulaire, *positionH* indique le point de coordonnée horizontale de l'image à afficher dans l'objet. La valeur doit être exprimée en pixels relativement à l'origine de l'image. Si le point de coordonnée horizontale est déjà visible dans l'objet, la commande ne fait rien (hormis si vous passez le second paramètre *, cf. ci-dessous).

Si vous passez le second paramètre optionnel * :

- la ligne rendue visible par la commande (si la liste a effectivement défilé) sera placée en première position de la liste. Si la ligne est située en fin de liste, cette option n'a pas d'effet.
- dans le contexte d'une image, les coordonnées demandées seront positionnées à l'origine de la variable image (0,0), même si ces coordonnées étaient déjà visibles dans l'objet.

Note : La commande **MARQUER ENREGISTREMENTS** comporte un paramètre * facultatif permettant de déléguer la gestion du défilement dans les formulaires à la commande **OBJET FIXER DEFILEMENT**.

Exemple 1

Cet exemple illustre la différence de fonctionnement de la commande avec une list box affichée en mode standard et hiérarchique :

```
OBJET FIXER DEFILEMENT(*;"malistbox";4;2;*) // afficher en tête la 4e ligne de la 2e colonne de la list box
```

Si cette instruction est appliquée à une list box affichée en mode standard :

France	Bretagne	Brest	120000	...
France	Bretagne	Quimper	80000	...
France	Bretagne	Rennes	200000	...
France	**Normandie**	Caen	220000	...
France	Normandie	Deauville	4000	...
France	Normandie	Cherbourg	41000	...
...	...	...	...	...

... les lignes et les colonnes de la list box défilent effectivement :

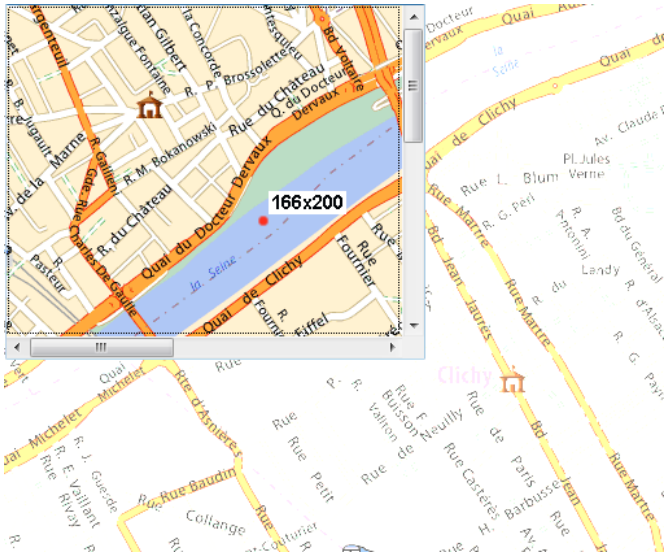
Normandie	Caen	220000	...	...
Normandie	Deauville	4000	...	...
Normandie	Cherbourg	41000	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

En revanche, si la même instruction est appliquée à la list box affichée en mode hiérarchique, les lignes défilent mais pas les colonnes car la 2e colonne appartient à la hiérarchie :

V **Normandie**	
Caen	220000
Deauville	4000
Cherbourg	41000
...	

Exemple 2

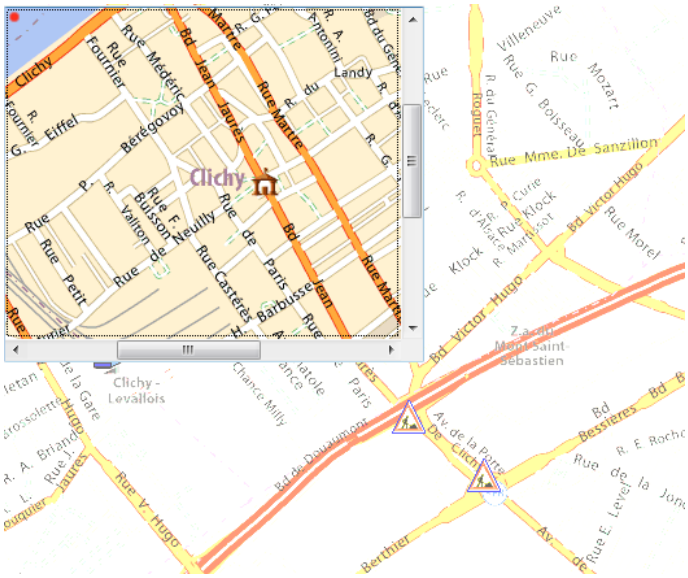
Vous souhaitez faire défiler une image incluse dans une variable de formulaire. Ce montage fait apparaître la partie visible de l'image ainsi que le point à afficher (166 pixels verticalement et 200 pixels horizontalement) :



Pour faire défiler la partie visible et afficher le point rouge à l'origine de la variable image, il vous suffit d'écrire :

```
OBJET FIXER DEFILEMENT(*;"maVar";166;200;*)
```

Vous obtenez le résultat suivant :



Attention dans ce cas, si vous omettez le second paramètre *, l'image ne défilera pas car le point défini est déjà visible.

OBJET FIXER EQUIVALENT CLAVIER

OBJET FIXER EQUIVALENT CLAVIER ({* ;} objet ; touche {; modifiers})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
touche	Chaîne	⇒	Touche à associer à l'objet
modifiers	Entier long	⇒	Masque ou combinaison de masques de touche(s) de modification

Description

La commande **OBJET FIXER EQUIVALENT CLAVIER** permet de définir ou de modifier dynamiquement l'équivalent clavier associé à l'objet ou aux objets désigné(s) par les paramètres *objet* et * pour le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable ou un champ. Dans ce cas, vous ne passez pas un nom mais une référence.

Passez dans le paramètre *touche* une chaîne indiquant la touche du clavier à associer à l'objet. Vous pouvez passer soit :

- un nom de touche standard, par exemple "B"
- une constante du thème **Touches équivalents clavier** (ou sa valeur) :

Constante	Type	Valeur	Comment
Raccourci avec Aide	Chaîne	[help]	
Raccourci avec Début	Chaîne	[home]	
Raccourci avec Echappement	Chaîne	[esc]	
Raccourci avec Effacement arrière	Chaîne	[backspace]	
Raccourci avec Entrée	Chaîne	[enter]	
Raccourci avec F1	Chaîne	[F1]	
Raccourci avec F10	Chaîne	[F10]	
Raccourci avec F11	Chaîne	[F11]	
Raccourci avec F12	Chaîne	[F12]	
Raccourci avec F13	Chaîne	[F13]	
Raccourci avec F14	Chaîne	[F14]	
Raccourci avec F15	Chaîne	[F15]	
Raccourci avec F2	Chaîne	[F2]	
Raccourci avec F3	Chaîne	[F3]	
Raccourci avec F4	Chaîne	[F4]	
Raccourci avec F5	Chaîne	[F5]	
Raccourci avec F6	Chaîne	[F6]	
Raccourci avec F7	Chaîne	[F7]	
Raccourci avec F8	Chaîne	[F8]	
Raccourci avec F9	Chaîne	[F9]	
Raccourci avec Fin	Chaîne	[end]	
Raccourci avec Flèche bas	Chaîne	[down arrow]	
Raccourci avec Flèche droite	Chaîne	[right arrow]	
Raccourci avec Flèche gauche	Chaîne	[left arrow]	
Raccourci avec Flèche haut	Chaîne	[up arrow]	
Raccourci avec Page préc	Chaîne	[page up]	
Raccourci avec Page suiv	Chaîne	[page down]	
Raccourci avec Retour Chariot	Chaîne	[return]	
Raccourci avec Suppression	Chaîne	[del]	
Raccourci avec Tabulation	Chaîne	[tab]	

Passez dans le paramètre *modifiers* une ou plusieurs touche(s) de modification à associer à la touche de raccourci. Pour définir le paramètre *modifiers*, passez une ou plusieurs des constante(s) de type "Masque" suivantes du thème **Evénements (Modifiers)** :

Constante	Type	Valeur	Comment
Masque touche commande	Entier long	256	Touche Ctrl sous Windows, touche Commande sous OS X
Masque touche contrôle	Entier long	4096	Touche Ctrl sous OS X, ou clic droit sous Windows et OS X
Masque touche majuscule	Entier long	512	Windows et OS X
Masque touche option	Entier long	2048	Touche Alt (aussi appelée Option sous OS X)

Note : Si vous omettez le paramètre *modifiers*, l'objet sera activé dès que vous appuierez sur la touche définie. Par exemple, si vous avez associé la touche "H" à un bouton, il sera activé dès que vous appuierez sur la touche H. Ce fonctionnement est à réserver à des interfaces spécifiques.

Exemple

Vous voulez associer un équivalent clavier différent en fonction de la langue courante de l'application. Dans l'événement sur chargement du formulaire, vous pouvez écrire :

```
Au cas ou
   :(vLang="FR")
```

```
      OBJET FIXER EQUIVALENT CLAVIER(*;"SortButton";"T";Masque touche commande+Masque touche majuscule)
  // Ctrl+Maj+T en français
    :(vLang="US")
      OBJET FIXER EQUIVALENT CLAVIER(*;"SortButton";"O";Masque touche commande+Masque touche majuscule)
  // Ctrl+Maj+O en anglais
Fin de cas
```

OBJET FIXER EVENEMENTS

OBJET FIXER EVENEMENTS ({* ;} objet ; tabEvénements ; mode)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet ou "" pour désigner le formulaire (si * est spécifié) ou Champ ou variable (si * est omis)
tabEvénements	Tableau entier long	⇒	Tableau d'événements à définir
mode	Entier long	⇒	Mode d'activation des événements définis dans tabEvénements

Description

La commande **OBJET FIXER EVENEMENTS** vous permet de modifier, pour le process courant, la configuration des événements formulaire du formulaire, de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).
Pour définir la configuration des événements du formulaire lui-même, passez le paramètre optionnel * et une chaîne vide "" dans *objet* : dans ce cas, vous désignez le formulaire courant.

Note : Si vous souhaitez modifier les événements d'un sous-formulaire lié à une table, seule la syntaxe basée sur le nom d'objet peut être utilisée.

Passez dans le paramètre *tabEvénements* un tableau Entier long contenant la liste des événements formulaire prédéfinis ou personnalisés que vous souhaitez modifier (le paramètre *mode* permet de préciser si la modification consiste à activer ou désactiver les événements). Pour désigner un événement prédéfini à modifier, vous pouvez passer dans chaque élément du tableau *tabEvénements* une des constantes suivantes, placées dans le thème "**Evénements formulaire**" :

Constante	Type	Valeur	Comment
Sur action suppression	Entier long	58	(*Listes hiérarchiques et List box*) L'utilisateur a demandé à supprimer un élément
Sur activation	Entier long	11	La fenêtre du formulaire passe au premier plan
Sur affichage corps	Entier long	8	Un enregistrement va être affiché dans la liste ou une ligne va être affichée dans la list box
Sur appel extérieur	Entier long	10	Le formulaire a reçu un appel de la commande **APPELER PROCESS**
Sur appel zone du plug in	Entier long	19	Un plug-in demande que sa méthode objet soit exécutée
Sur après frappe clavier	Entier long	28	Un caractère vient d'être saisi dans l'objet qui a le focus. **Lire texte edite** retourne le contenu avec ce caractère.
Sur après modification	Entier long	45	Le contenu de l'objet saisissable qui a le focus vient d'être modifié
Sur après tri	Entier long	30	(*List box uniquement*) Un tri standard vient d'être effectué dans une colonne de list box
Sur avant frappe clavier	Entier long	17	Un caractère vient d'être saisi dans l'objet qui a le focus. **Lire texte edite** retourne le contenu sans ce caractère
Sur avant saisie	Entier long	41	(*List box uniquement*) Une cellule de list box est sur le point de passer en mode édition
Sur bouton barre outils Mac	Entier long	55	L'utilisateur a cliqué sur le bouton de gestion de la barre d'outils sous Mac OS
Sur case de fermeture	Entier long	22	On a cliqué sur la case de fermeture de la fenêtre
Sur chargement ligne	Entier long	40	En mode saisie en liste, un enregistrement est chargé en modification (l'utilisateur a cliqué sur la ligne de l'enregistrement et un champ passe en édition)
Sur chargement ressource URL	Entier long	48	(*Zones Web uniquement*) Une nouvelle ressource est chargée dans la zone Web
Sur clic	Entier long	4	Un clic est survenu sur un objet
Sur clic entête	Entier long	42	(*List box uniquement*) Un clic est survenu dans l'en-tête d'une colonne de list box
Sur clic flèche	Entier long	38	(*Boutons 3D uniquement*) La zone "flèche" d'un bouton 3D reçoit un clic
Sur clic long	Entier long	39	(*Boutons 3D uniquement*) Un bouton 3D reçoit un clic et le bouton de la souris reste enfoncé pendant un certain laps de temps
Sur clic pied	Entier long	57	(*List box uniquement*) Un clic est survenu dans le pied d'une list box ou d'une colonne de list box
Sur contracter	Entier long	44	(*Listes hiérarchiques et list box hiérarchiques*) Un élément de liste hiérarchique ou de list box hiérarchique a été contracté via un clic ou une touche du clavier
Sur début chargement URL	Entier long	47	(*Zones Web uniquement*) Un nouvel URL est chargé dans la zone Web
Sur début glisser	Entier long	46	Un objet est en cours de glisser
Sur début survol	Entier long	35	Le curseur de la souris entre dans la zone graphique d'un objet
Sur défilement image	Entier long	59	*Variables ou champs image et List Box* : L'utilisateur fait défiler le contenu du champ image, de la variable image ou de la list box à l'aide de la souris ou d'une touche du clavier.

Constante	Type	Valeur	Comment
Sur déplacement colonne	Entier long	32	(*List box uniquement*) Une colonne de list box est déplacée par l'utilisateur via le glisser-déposer
Sur déplacement ligne	Entier long	34	(*List box uniquement*) Une ligne de list box est déplacée par l'utilisateur via le glisser-déposer
Sur déployer	Entier long	43	(*Listes hiérarchiques et List box hiérarchiques*) Un élément de liste hiérarchique ou de list box hiérarchique a été déployé via un clic ou une touche du clavier
Sur déposer	Entier long	16	Des données sont déposées sur un objet
Sur désactivation	Entier long	12	La fenêtre du formulaire passe en arrière-plan
Sur données modifiées	Entier long	20	Les données d'un objet ont été modifiées
Sur double clic	Entier long	13	Un double-clic est survenu sur un objet
Sur entête	Entier long	5	L'en-tête du formulaire va être imprimé ou affiché
Sur erreur chargement URL	Entier long	50	(*Zones Web uniquement*) Une erreur s'est produite durant le chargement de l'URL
Sur fermeture corps	Entier long	26	Le formulaire détaillé se referme et on retourne au formulaire sortie
Sur filtrage URL	Entier long	51	(*Zones Web uniquement*) Un URL a été bloqué par la zone Web
Sur fin chargement URL	Entier long	49	(*Zones Web uniquement*) Toutes les ressources de l'URL ont été chargées
Sur fin survol	Entier long	36	Le curseur de la souris sort de la zone graphique d'un objet
Sur gain focus	Entier long	15	Un objet de formulaire prend le focus
Sur glisser	Entier long	21	Des données sont glissées sur un objet
Sur impression corps	Entier long	23	Le corps du formulaire va être imprimé
Sur impression pied de page	Entier long	7	Le pied de page du formulaire va être imprimé
Sur impression sous total	Entier long	6	Une rupture du formulaire va être imprimée
Sur libération	Entier long	24	Le formulaire se referme et est déchargé
Sur menu sélectionné	Entier long	18	Une commande de menu a été sélectionnée
Sur minuteur	Entier long	27	Le nombre de ticks défini par **FIXER MINUTEUR** est atteint
Sur modif variable liée	Entier long	54	La variable liée à un sous-formulaire est modifiée.
Sur nouvelle sélection	Entier long	31	• *List box* : la sélection courante de lignes ou de colonnes est modifiée • *Enregistrements en liste* : l'enregistrement courant ou la sélection courante de lignes est modifié(e) dans un formulaire en liste ou un sous-formulaire • *Liste hiérarchique* : la sélection dans la liste est modifiée à la suite d'un clic ou de la frappe d'une touche au clavier • *Variable ou champ saisissable* : la sélection de texte ou la position du curseur dans la zone est modifiée à la suite d'un clic ou de la frappe d'une touche au clavier
Sur ouverture corps	Entier long	25	Le formulaire détaillé associé au formulaire sortie ou à la listbox est sur le point d'être ouvert
Sur ouverture lien externe	Entier long	52	(*Zones Web uniquement*) Un URL externe a été ouvert dans le navigateur
Sur perte focus	Entier long	14	Un objet de formulaire perd le focus
Sur redimensionnement	Entier long	29	La fenêtre du formulaire est redimensionnée
Sur redimensionnement colonne	Entier long	33	(*List box uniquement*) La largeur d'une colonne de list box est modifiée par l'utilisateur via la souris
Sur refus ouverture fenêtre	Entier long	53	(*Zones Web uniquement*) Une fenêtre pop up a été bloquée
Sur survol	Entier long	37	Le curseur de la souris a bougé d'au moins un pixel OU une touche de modification (Ctrl, Alt, Verr Maj.) a été enfoncée. Si l'événement est coché pour un objet uniquement, il n'est généré que lorsque le curseur se trouve dans la zone graphique de l'objet
Sur validation	Entier long	3	La saisie des données dans l'enregistrement est validée

Il est important de noter que l'événement Sur chargement est absent de cette liste : il ne peut pas être défini, car lors de l'exécution de la commande il a déjà été généré.

Vous pouvez également passer dans *tabEvénements* toute valeur correspondant à un événement personnalisé. Dans ce cas, il est recommandé d'utiliser des valeurs négatives (cf. commande **APPELER CONTENEUR SOUS FORMULAIRE**).

Le paramètre *mode* vous permet de définir le traitement global à effectuer pour les éléments du tableau. Pour cela, vous pouvez passer une des constantes suivantes, placées dans le thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Activer événements autres inchangés	Entier long	1	Tous les événements listés dans le tableau *tabEvénements* sont activés, le statut des autres événements est inchangé
Activer événements inactiver autres	Entier long	0	Tous les événements listés dans le tableau *tabEvénements* sont activés, tous les autres événements sont désactivés
Inactiver événements autres inchangés	Entier long	2	Tous les événements listés dans le tableau *tabEvénements* sont désactivés, le statut des autres événements est inchangé

La commande **OBJET FIXER EVENEMENTS** peut entraîner l'activation d'événements non pris en charge par l'*objet* (en fonction de son type). Dans ce cas, les événements seront simplement ignorés.

Si un *objet* est dupliqué après l'appel de la commande **OBJET FIXER EVENEMENTS**, la configuration résultante d'activation/désactivation de chaque événement est également dupliquée.

Exemple 1

Activation de trois événements formulaire pour un ensemble d'objets list box, et désactivation des autres événements :

```
TABLEAU ENTIER LONG($MyEventsOnLB;3)
$MyEventsOnLB {1}:=Sur après tri
$MyEventsOnLB {2}:=Sur déplacement colonne
$MyEventsOnLB {3}:=Sur redimensionnement colonne
OBJET FIXER EVENEMENTS(*;"MesLB@";$MyEventsOnLB;Activer événements inactiver autres)
     // active 3 événements et désactive tous les autres
```

Exemple 2

Désactivation de trois événements formulaire pour un ensemble d'objets list box, sans modifier les autres événements :

```
TABLEAU ENTIER LONG($MyEventsOnLB;3)
$MyEventsOnLB {1}:=Sur après tri
$MyEventsOnLB {2}:=Sur déplacement colonne
$MyEventsOnLB {3}:=Sur redimensionnement colonne
OBJET FIXER EVENEMENTS(*;"MesLB@";$MyEventsOnLB;Inactiver événements autres inchangés)
    // désactive uniquement les 3 événements
```

Exemple 3

Activation d'un événement formulaire pour un objet, sans modifier les autres événements :

```
TABLEAU ENTIER LONG($MyEventsOnLB;1)
$MyEventsOnLB {1}:=Sur déplacement colonne
OBJET FIXER EVENEMENTS(*;"Col1";$MyEventsOnLB;Activer événements autres inchangés)
    // active uniquement l'événement
```

Exemple 4

Désactivation de tous les événements du formulaire :

```
TABLEAU ENTIER LONG($MyFormEvents;0)
OBJET FIXER EVENEMENTS(*;"";$MyFormEvents;Activer événements inactiver autres)
    // désactive tous les événements
```

Exemple 5

Désactivation d'un seul événement du formulaire sans modifier les autres :

```
TABLEAU ENTIER LONG($MyFormEvents;1)
$MyFormEvents{1}:=Sur minuteur
OBJET FIXER EVENEMENTS(*;"";$MyFormEvents;Inactiver événements autres inchangés)
    // désactive uniquement l'événement
```

OBJET FIXER FEUILLE DE STYLE

OBJET FIXER FEUILLE DE STYLE ({* ;} objet ; nomFeuilleStyle)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
nomFeuilleStyle	Texte	⇒	Nom de la feuille de style

Description

La commande **OBJET FIXER FEUILLE DE STYLE** vous permet de modifier, pour le process courant, la feuille de style associée à l'objet ou aux objets désigné(s) par les paramètres *objet* et *. Une feuille de style modifie la police, la taille de police et (hormis pour les feuilles de style automatique) le style de police.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans le paramètre *nomFeuilleStyle* le nom de la feuille de style à appliquer à l'*objet*. Vous pouvez également passer soit :

- un nom de feuille de style existante (si la feuille de style n'existe pas, une erreur est retournée, que vous pouvez intercepter l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**),
- une chaîne vide ("") pour ne pas appliquer de feuille de style à l'*objet*.
- une des constantes suivantes du thème "**Styles de caractères**" pour appliquer une feuille de style automatique :

Constante	Type	Valeur	Comment
Feuille de style automatique	Chaîne	__automatic__	Utilisée par défaut pour tous les objets
Feuille de style automatique_additionnel	Chaîne	__automatic_additional_text__	Prise en charge par les textes statiques, champs et variables uniquement. Utilisée pour du texte additionnel dans les boîtes de dialogue.
Feuille de style automatique_texte principal	Chaîne	__automatic_main_text__	Prise en charge par les textes statiques, champs et variables uniquement. Utilisée pour le texte principal des boîtes de dialogue.

Si une feuille de style avait déjà été associée à l'objet en mode Développement, l'appel de cette commande la remplace pour le process courant.

Si vous utilisez au cours de la session les commandes **ST FIXER ATTRIBUTS**, **ST FIXER TEXTE** ou **OBJET FIXER POLICE** sur l'*objet* afin de modifier sa police ou sa taille de police, la référence à la feuille de style est automatiquement supprimée de l'objet -- même si vous affectez des attributs identiques à ceux de la feuille de style. En revanche, si vous modifiez le style (gras, italique...), par exemple avec les commandes **ST FIXER ATTRIBUTS** ou **OBJET FIXER STYLE POLICE**, ces nouvelles propriétés s'ajoutent à la feuille de style pour la durée de la session.

OBJET FIXER FILTRE SAISIE

OBJET FIXER FILTRE SAISIE ({* ;} objet ; filtreSaisie)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
filtreSaisie	Chaîne	⇒	Nouveau filtre de saisie pour la zone saisissable

Description

OBJET FIXER FILTRE SAISIE remplace le filtre de saisie pour *objet* par *filtreSaisie* dans le formulaire courant affiché à l'écran.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

La commande **OBJET FIXER FILTRE SAISIE** peut être utilisée dans des formulaires entrée et des dialogues et peut être appliquée aux champs et variables saisissables acceptant les filtres de saisie en mode Développement.

Pour enlever un filtre, passez une chaîne vide dans le paramètre *filtreSaisie*.

Note : Cette commande ne peut pas être utilisée avec des champs situés dans le formulaire "liste" d'un sous-formulaire.

Note : Pour pouvoir exploiter les filtres de saisie que vous avez créés dans la Boîte à outils, préfixez le nom du filtre, dans le paramètre *filtreSaisie*, d'une barre verticale (|).

Exemple 1

L'exemple suivant définit le filtre de saisie pour le champ code postal. Si l'adresse se trouve en France, le filtre est paramétré pour les codes postaux français. Sinon, le filtre peut accepter toute valeur saisie :

```
Si(Pays="France") ` Fixer le filtre au format du code postal français
   OBJET FIXER FILTRE SAISIE([Sociétés]Code postal;"&#####")
Sinon ` Fixer le filtre pour qu'il accepte toute valeur alphanumérique
   OBJET FIXER FILTRE SAISIE([Sociétés]Code postal;"~@")
Fin de si
```

Exemple 2

L'exemple suivant autorise uniquement la saisie des lettres “a”, “b”, “c” ou “g” dans un champ comportant deux lettres :

```
OBJET FIXER FILTRE SAISIE([Table]Champ;"&"+Caractere(Guillemets)+"a;b;c;g"+Caractere(Guillemets)+"##")
```

Note : Cet exemple définit le filtre de saisie *&"a;b;c;g"##*.

OBJET FIXER FORMATAGE

OBJET FIXER FORMATAGE ({* ;} objet ; formatAffich)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
formatAffich	Chaîne	⇒	Nouveau format d'affichage de l'objet

Description

OBJET FIXER FORMATAGE remplace le format d'affichage du ou des objet(s) spécifié(s) par *objet* avec le format que vous avez passé dans *formatAffich*. Le nouveau format est utilisé uniquement pour l'affichage courant, il n'est pas stocké avec le formulaire.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

La commande **OBJET FIXER FORMATAGE** peut être indifféremment utilisée dans des formulaires entrée ou sortie (affichés ou imprimés) et appliquée aux champs ou aux variables (saisissables ou non saisissables). Bien entendu, vous devez utiliser un format d'affichage compatible avec le type de données présentes dans l'objet ou avec l'objet lui-même.

Booléens

Pour formater des champs booléens, vous disposez de deux possibilités :

- vous pouvez passer une valeur simple dans *formatAffich*. Dans ce cas, le champ sera affiché sous forme de case à cocher, son libellé sera la valeur définie.
- vous pouvez passer deux valeurs séparées par un point-virgule (;) dans *formatAffich*. Dans ce cas, le champ sera affiché sous forme de deux boutons radio.

Dates

Pour formater des champs ou variables de type Date, passez **Caractere(n)** dans *formatAffich*, où *n* peut être une des constantes prédéfinies suivantes (thème **Formats d'affichage des dates**) :

Constante	Type	Valeur	Comment
Date RFC 1123	Entier long	10	Fri, 10 Sep 2010 13:07:20 GMT
Interne date abrégé	Entier long	6	6 déc 1996
Interne date court	Entier long	7	06/12/2006
Interne date court spécial	Entier long	4	06/12/06 (mais 06/12/1896 ou 06/12/2096)
Interne date long	Entier long	5	6 décembre 2006
ISO Date	Entier long	8	2006-06-12T00:00:00 (format obsolète)
ISO Date GMT	Entier long	9	2010-09-13T16:11:53Z
Système date abrégé	Entier long	2	mer. 25 déc. 2006
Système date court	Entier long	1	06/12/2006
Système date long	Entier long	3	mercredi 6 décembre 2006
Vide si date nulle	Entier long	100	"" au lieu de 0 en cas de valeur nulle. Cette constante doit être additionnée au format d'affichage.

Note : La constante Vide si date nulle doit être additionnée au format, elle indique qu'en cas de valeur nulle 4D doit afficher une zone vide au lieu de zéros.

Heures

Pour formater des champs ou variables de type Heure, passez **Caractere(n)** dans *formatAffich*, où *n* peut être une des constantes prédéfinies suivantes (thème **Formats d'affichage des heures**) :

Constante	Type	Valeur	Comment
h mn	Entier long	2	01:02
h mn Matin Après Midi	Entier long	5	1:02 du matin
h mn s	Entier long	1	01:02:03
Heures minutes	Entier long	4	1 heure 2 minutes
Heures minutes secondes	Entier long	3	1 heure 2 minutes 3 secondes
ISO heure	Entier long	8	0000-00-00T01:02:03
Minutes secondes	Entier long	7	62 minutes 3 secondes
mn s	Entier long	6	62:03
Système heure court	Entier long	9	01:02:03
Système heure long	Entier long	11	1:02:03 AM HNEC (Mac uniquement)
Système heure long abrégé	Entier long	10	1•02•03 AM (Mac uniquement)
Vide si heure nulle	Entier long	100	"" au lieu de 0

Note : La constante Vide si heure nulle doit être additionnée au format, elle indique qu'en cas de valeur nulle 4D doit afficher une zone vide au lieu de zéros.

Images

Pour formater des champs ou variables de type Image, passez **Caractere(n)** dans *formatAffich*, où *n* peut être une des constantes prédéfinies suivantes (thème **Formats d'affichage des images**) :

Constante	Type	Valeur
Mosaïque	Entier long	7
Non tronquée	Entier long	2
Proportionnelle	Entier long	5
Proportionnelle centrée	Entier long	6
Sur fond	Entier long	3
Tronquée centrée	Entier long	1
Tronquée non centrée	Entier long	4

Alphas et numériques

Pour formater des champs ou variables de type alpha ou numérique, passez directement le libellé du format dans le paramètre *formatAffich*.

Pour plus d'informations sur les formats d'affichage, reportez-vous aux sections **Formats numériques** et **Formats alphanumériques** dans le manuel *Mode Développement* de 4D.

Note : Pour pouvoir exploiter les formats d'affichage personnalisés que vous avez créés dans la boîte à outils, préfixez le nom du format, dans le paramètre *formatAffich*, d'une barre verticale (|).

Boutons image

Pour formater des boutons image, passez dans le paramètre *formatAffich* une chaîne de caractères respectant la syntaxe suivante :
cols;lignes;image;mode{;ticks}

- *cols* = nombre de colonnes de l'image
- *lignes* = nombre de lignes de l'image
- *image* = image utilisée, provenant de la bibliothèque d'images ou d'une variable image :
  - si l'image provient de la bibliothèque d'images, saisissez son numéro, précédé d'un point d'interrogation (ex. : "?250").
  - si l'image provient d'une variable image, saisissez le nom de la variable.
- *mode* = mode d'affichage et de fonctionnement du bouton image. Ce paramètre peut prendre les valeurs 0, 1, 2, 16, 32, 64 et 128, chaque valeur représentant un mode d'affichage ou de fonctionnement. Ces valeurs sont cumulatives ; en d'autres termes, pour sélectionner les valeurs 64 et 1, passez 65 dans le paramètre *mode*. Voici le détail de chaque valeur :
  - *mode* = 0 (pas d'option)
    Affiche l'image suivante de la série lorsque l'utilisateur clique sur le bouton. Affiche l'image précédente de la série lorsque l'utilisateur effectue Maj+clic sur le bouton. La séquence d'images s'arrête lorsqu'on atteint la dernière image de la série. En d'autres termes, le bouton ne retourne pas à la première image de la série.
  - *mode* = 1 (Défilement continu sur clic)
    Similaire au précédent, à la différence près que lorsque l'utilisateur clique sur l'image et maintient le bouton de la souris enfoncé, l'enchaînement des images est continu (c'est-à-dire que la série défile comme une animation). La séquence d'images s'arrête lorsqu'on atteint la dernière image de la série.
  - *mode* = 2 (Recommencer la séquence)
    Similaire au précédent, à la différence près que le défilement des images est "rebouclé" lorsqu'on atteint la dernière image de la séquence de défilement : une fois la dernière image atteinte, la première image est de nouveau affichée et la séquence recommence.
  - *mode* = 16 (Bascule sur passage du curseur)
    Le contenu du bouton image est modifié lorsque le curseur de la souris passe au-dessus de lui, sans que l'utilisateur ne clique. L'image initiale est rétablie lorsque le curseur quitte la zone du bouton. Ce mode, aussi appelé "Roll over", est fréquemment utilisé dans les navigateurs Web et dans les applications multimedia. L'image affichée est la dernière du tableau d'imagettes, sauf si le mode 128 (Dernière imagette si désactivé) est également sélectionné — dans ce cas, c'est l'avant-dernière imagette qui est utilisée comme "bascule".
  - *mode* = 32 (Retour sur relâchement du clic)
    Ce mode fonctionne avec deux images ; il indique que le bouton doit toujours afficher la première image, sauf quand l'utilisateur clique dessus. En d'autres termes, le bouton affiche l'image A par défaut, l'image B lorsqu'il reçoit un clic souris, et de nouveau l'image A dès que le bouton de la souris est relâché. Ce mode permet de réaliser un bouton d'action avec une image différente pour chaque état (normal et enfoncé). Vous pouvez ainsi créer un effet 3D personnalisé ou toute image symbolisant l'action effectuée par bouton.
  - *mode* = 64 (Transparent)
    Permet de rendre transparent le fond de l'image.
  - *mode* = 128 (Dernière imagette si désactivé)
    Permet d'indiquer que la dernière image de la série doit être utilisée lorsque le bouton est inactivé. Avec ce paramétrage, 4D affiche la dernière "partie" de l'image référencée lorsque le bouton image est inactivé. L'image d'inactivation est traitée à part par 4D : lorsque vous combinez cette option avec les valeurs 0, 1 ou 2 dans le paramètre *mode*, la dernière image est exclue de la séquence associée au bouton et n'apparaîtra que lorsqu'il sera inactivé.
- *ticks* = activation du mode "défilement automatique tous les N ticks" et intervalle de temps séparant l'affichage de chaque image. Ce paramètre optionnel, s'il est passé, provoque le défilement automatique et en boucle du contenu du bouton image à la vitesse spécifiée. Par exemple, si vous passez "2;3;?16807;0;10", la variation du bouton image s'effectuera tous les 10 ticks. Dans ce mode, toutes les autres options sont ignorées — à l'exception de l'option "Transparent" (mode 64).

Pop up menus image

Pour formater des pop up menus image, passez dans le paramètre *formatAffich* une chaîne de caractères respectant la syntaxe suivante :
cols;lignes;image;margeH;margeV;mode

- *cols* = nombre de colonnes de l'image
- *lignes* = nombre de lignes de l'image
- *image* = image utilisée, provenant de la bibliothèque d'images ou d'une variable image :
  - si l'image provient de la bibliothèque d'images, saisissez son numéro, précédé d'un point d'interrogation (ex. : "?250"),
  - si l'image provient d'une variable image, saisissez le nom de la variable.
- *margeH* = marge en pixels entre les limites horizontales du menu et l'image.
- *margeV* = marge en pixels entre les limites verticales du menu et l'image.
- *mode* = mode de transparence du pop up menu image. Accepte les valeurs 0 et 64 :
  - *mode* = 0 : le pop-up menu image n'est pas transparent,
  - *mode* = 64 : le pop-up menu image est transparent.

Thermomètres et règles

Pour formater des objets de type thermomètre ou règle, passez dans le paramètre *formatAffich* une chaîne de caractères respectant la syntaxe suivante :
min;max;unité;pas;mode{;format{;affichage}}

- *min* = valeur de la graduation d'origine de la jauge
- *max* = valeur de la graduation de fin de la jauge
- *unité* = intervalle entre les graduations de la jauge
- *pas* = intervalle de déplacement du curseur dans la jauge
- *mode* = mode d'affichage et de fonctionnement de la jauge. Ce paramètre accepte les valeurs 0, 2, 3, 16, 32 et 128. Ces valeurs peuvent être cumulées afin de définir plusieurs options (hormis le mode 128). Voici le détail de chaque valeur :
  - *mode* = 0 : ne pas afficher les libellés
  - *mode* = 2 : afficher les libellés à droite ou au-dessous de la jauge
  - *mode* = 3 : afficher les libellés à gauche ou au-dessus de la jauge
  - *mode* = 16 : afficher les graduations en regard des libellés
  - *mode* = 32 : déclencher la méthode objet avec l'événement Sur données modifiées pendant que l'utilisateur change la valeur de la jauge. Par défaut, la méthode est exécutée après la modification.
  - *mode* = 128: activer le mode "Barber shop" (animation continue). Cette valeur ne peut pas être cumulée. Dans ce mode, les autres paramètres sont ignorés (hormis le paramètre *affichage* s'il est passé). Pour plus d'informations sur ce mode, reportez-vous au manuel Mode Développement.
- *format* = format d'affichage des graduations de la jauge.
  A noter les libellés et les graduations sont automatiquement masqués si la taille de l'objet jauge ne permet pas de les afficher correctement.
- *affichage* = options d'affichage spécifiques. Dans le cas des thermomètres, ce paramètre n'est pris en compte que si le sous-paramètre *mode* vaut 128.
  - *affichage* = 0 (ou est omis) : afficher une règle standard / afficher un thermomètre en animation continue "barber shop".
  - *affichage* = 1 : activer le mode "Stepper" pour une règle / activer le mode "Progression asynchrone" pour un thermomètre. Pour plus d'informations sur ces options, reportez-vous au manuel *Mode Développement*.

Cadrans

Pour formater des objets de type cadran, passez dans le paramètre *formatAffich* une chaîne de caractères respectant la syntaxe suivante :
min;max;unité;pas{;mode}

- *min* = valeur de la graduation d'origine du cadran
- *max* = valeur de la graduation de fin du cadran
- *unité* = intervalle entre les graduations du cadran
- *pas* = intervalle de déplacement du curseur dans le cadran
- *mode* = mode de fonctionnement du cadran (facultatif). Ce paramètre accepte uniquement la valeur 32 : déclencher la méthode objet avec l'événement Sur données modifiées pendant que l'utilisateur change la valeur du cadran. Par défaut, la méthode est exécutée après la modification.

Grilles de boutons

Pour formater des grilles de boutons, passez dans le paramètre *formatAffich* une chaîne de caractères respectant la syntaxe suivante :
cols;lignes

- *cols* = nombre de colonnes de la grille
- *lignes* = nombre de lignes de la grille

Note : Pour plus d'informations sur les formats d'affichage des objets de formulaire, reportez-vous au manuel Mode Développement de 4D.

Boutons 3D

Pour formater des boutons 3D, passez dans le paramètre *formatAffich* une chaîne de caractères respectant la syntaxe suivante :
titre;image;imageFond;posTitre;titreVisible;icôneVisible;style;margeHor;margeVert;décalageIcône;popupMenu;hyperlien;nbEtats

- *titre* = titre du bouton. Cette valeur peut être exprimée sous forme de texte ou de numéro de ressource (ex. : ":16800,1")
- *image* = image associée au bouton, provenant de la bibliothèque d'images, d'une variable image ou d'un fichier image :
  - si l'image provient de la bibliothèque d'images, saisissez son numéro, précédé d'un point d'interrogation (ex. : "?250").
  - si l'image provient d'une variable image, saisissez le nom de la variable.
  - si l'image provient d'un fichier stocké dans le dossier Resources de la base, saisissez un URL du type "#{dossier/}nomimage" ou "file:{dossier/}nomimage".
- *imageFond* = image de fond associée au bouton (style Personnalisé), provenant de la bibliothèque d'images, d'une variable image ou d'un fichier stocké dans le dossier Resources (cf. ci-dessus).
- *posTitre* = position du titre du bouton. Cinq valeurs sont possibles :
  - *posTitre* = 1 : Gauche
  - *posTitre* = 2 : Haut
  - *posTitre* = 3 : Droite
  - *posTitre* = 4 : Bas
  - *posTitre* = 5 : Centre
- *titreVisible* = Titre visible ou non. Deux valeurs sont possibles :
  - *titreVisible* = 0 : le titre est masqué
  - *titreVisible* = 1 : le titre est affiché
- *icôneVisible* = Icône visible ou non. Deux valeurs sont possibles :
  - *icôneVisible* = 0 : l'icône est masquée
  - *icôneVisible* = 1 : l'icône est affichée
- *style* = Style du bouton. La valeur de cette option détermine la prise en compte de certaines autres options (par exemple imageFond). Les valeurs de style suivantes sont possibles :
  - *style* = 0 : Aucun
  - *style* = 1 : Décalage du fond
  - *style* = 2 : Bouton poussoir
  - *style* = 3 : Bouton barre outils
  - *style* = 4 : Personnalisé
  - *style* = 5 : Rond
  - *style* = 6 : Petit carré système
  - *style* = 7 : Office XP
  - *style* = 8 : Bevel
  - *style* = 9 : Bevel arrondi
  - *style* = 10 : Contracter/Déployer
  - *style* = 11 : Aide

    - *style* = 12 : OS X Textured
    - *style* = 13 : OS X Gradient
- *margeHor* = Marge horizontale. Nombre de pixels délimitant les marges internes à droite et à gauche du bouton (zones dans lesquelles l'icône et le texte ne doivent pas empiéter).
- *margeVert* = Marge verticale. Nombre de pixels délimitant les marges internes en haut et en bas du bouton (zones dans lesquelles l'icône et le texte ne doivent pas empiéter).
- *décalageIcône* = Décalage de l'icône vers la droite et le bas. Cette valeur, exprimée en pixels, indique le décalage de l'icône du bouton vers la droite et le bas en cas de clic (la même valeur est utilisée pour les deux directions).
- *popupMenu* = Association d'un pop up menu au bouton. Trois valeurs sont possibles :
    - *popupMenu* = 0 : Sans pop up menu
    - *popupMenu* = 1 : Avec pop up menu lié
    - *popupMenu* = 2 : Avec pop up menu séparé
- *hyperlien* = Soulignement du titre sur le passage du curseur de la souris pour évoquer un lien hypertexte (mécanisme obsolète). Deux valeurs sont possibles :
    - *hyperlien* = 0 : le titre n'est pas souligné au passage de la souris
    - *hyperlien* = 1 : le titre est souligné au passage de la souris
- *nbEtats* = Nombre d'états présents dans l'image utilisée comme icône pour le bouton 3D, et qui seront utilisées par 4D pour représenter les états standard du bouton (de 0 à 4).

Certaines options ne sont pas prises en charge dans tous les styles de boutons 3D. De plus, dans certains cas vous pourrez souhaiter ne pas modifier toutes les options. Pour ne pas passer une option, il suffit d'omettre la valeur correspondante. Par exemple, pour ne pas passer les options *titreVisible*, *margeVert* et *hyperlien*, vous pouvez écrire :

```
OBJET FIXER FORMATAGE(maVar;"JoliBouton;?256;:562;1;;1;4;5;;5;0;;2")
```

En-têtes de list box

Pour formater l'icône d'un en-tête de list box, passez dans le paramètre *formatAffich* une chaîne de caractères respectant la syntaxe suivante : *image;posIcone*

- *image* = image de l'en-tête, provenant de la bibliothèque d'images, d'une variable image ou d'un fichier image :
    - si l'image provient de la bibliothèque d'images, saisissez son numéro, précédé d'un point d'interrogation (ex. : "?250").
    - si l'image provient d'une variable image, saisissez le nom de la variable.
    - si l'image provient d'un fichier stocké dans le dossier Resources de la base, saisissez un URL du type "#{dossier/}nomimage" ou "file:{dossier/}nomimage".
- *posIcone* = position de l'icône dans l'en-tête. Deux valeurs sont possibles :
    - *posIcone* = 1 : gauche
    - *posIcone* = 2 : droite

Cette fonctionnalité est utile par exemple lorsque vous voulez travailler avec une icône de tri personnalisée.

Exemple 1

La ligne de code suivante formate le champ *[Employés]Date embauche* au cinquième format de date.

```
OBJET FIXER FORMATAGE([Employés]Date embauche;Caractere(Interne date long))
```

Exemple 2

L'exemple suivant change le format d'un champ *[Sociétés]Code postal* selon la longueur du code postal :

```
Si(Longueur([Sociétés]Code postal)=9)
   OBJET FIXER FORMATAGE([Sociétés]Code postal;"#####-####")
Sinon
   OBJET FIXER FORMATAGE([Sociétés]Code postal;"#####")
Fin de si
```

Exemple 3

L'exemple suivant formate la valeur d'un champ numérique selon qu'elle est positive, négative ou nulle :

```
OBJET FIXER FORMATAGE([Stats]Résultat;"### ##0.00;(### ##0.00);")
```

Exemple 4

L'exemple suivant définit le format d'un champ booléen pour afficher soit "Marié" soit "Célibataire" au lieu des valeurs par défaut "Oui" et "Non" :

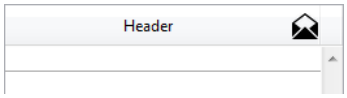
```
OBJET FIXER FORMATAGE([Employés]Situation;"Marié;Célibataire")
```

Exemple 5

En supposant que vous avez stocké un fichier image nommé "envelope_open.png" dans le dossier Resources de la base, vous pouvez écrire :

```
vIcon:="#envelope_open.png"
```

```
vPos:="2" // Droite
OBJET FIXER FORMATAGE(*;"Header1";vIcon+";"+vPos)
```



Exemple 6

L'exemple suivant définit le format d'un champ booléen pour afficher une case à cocher libellée "Classé" :

```
OBJET FIXER FORMATAGE([Dossier]Classement;"Classé")
```

Exemple 7

Vous disposez d'un tableau d'imagettes contenant 1 ligne et 4 colonnes, destiné à afficher un bouton image ("actif par défaut", "bouton cliqué", "survol du curseur" et "inactivé"). Vous souhaitez lui associer les options Bascule sur passage du curseur, Retour sur relâchement du clic et Dernière imagette si désactivé :

```
OBJET FIXER FORMATAGE(*;"BoutonImage";"4;1;?15000;176")
```

Exemple 8

Passage d'un thermomètre en mode "Barber shop" :

```
OBJET FIXER FORMATAGE($Monthermo;";;;;128")
$Monthermo:=1 `Déclencher l'animation
```

OBJET FIXER IMPRESSION TAILLE VARIABLE

OBJET FIXER IMPRESSION TAILLE VARIABLE ({* ;} objet ; tailleVariable {; fixeSousForm})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
tailleVariable	Booléen	⇒	Vrai = Impression taille variable, Faux = Impression taille fixe
fixeSousForm	Entier long	⇒	Options d'impression en taille fixe des sous-formulaires

Description

La commande **OBJET FIXER IMPRESSION TAILLE VARIABLE** vous permet de modifier la propriété d'impression en taille variable de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

La propriété d'impression en taille variable est disponible pour les objets suivants :

- variables et champs de type Texte et Image (cf. paragraphe **Impression taille variable** dans le manuel *Mode Développement*)
- zones 4D Write Pro (cf. section **Utiliser une zone 4D Write Pro** du manuel de référence de 4D Write Pro).
- sous-formulaires. Les sous-formulaires disposent d'une option supplémentaire pour l'impression en taille fixe (cf. paragraphe **Impression du sous-formulaire** dans le manuel *Mode Développement*) ; la commande permet de configurer cette option via le paramètre *fixeSousForm*.

Si vous appliquez cette commande à un objet ne prenant pas en charge cette propriété, la commande ne fait rien.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez un booléen dans le paramètre *tailleVariable* : si vous passez **Vrai**, l'objet sera imprimé en taille variable. Si vous passez **Faux**, il sera imprimé en taille fixe.

Le paramètre optionnel *fixeSousForm* vous permet de définir une option supplémentaire lorsque vous avez passé **Faux** dans le paramètre *tailleVariable* et que *objet* est un sous-formulaire (il est ignoré dans tous les autres cas). Dans ce cas, vous pouvez définir le mode d'impression en taille fixe du sous-formulaire. Vous pouvez passer une des constantes suivantes, placées dans le thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Impression limitée avec report	Entier long	2	La taille initiale de la zone du sous-formulaire est conservée mais 4D imprime le formulaire plusieurs fois pour imprimer tous les enregistrements.
Impression limitée par le cadre	Entier long	1	4D n'imprime que les enregistrements qui apparaissent dans la zone du sous-formulaire. Le formulaire n'est imprimé qu'une fois et les enregistrements qui ne sont pas imprimés sont ignorés.

OBJET FIXER LISTE PAR NOM

OBJET FIXER LISTE PAR NOM ({* ;} objet {; typeListe}; énumération)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
typeListe	Entier long	⇒	Type de liste : Liste énumération, Liste obligations ou Liste exclusions
énumération	Chaîne	⇒	Nom de l'énumération (définie en mode Développement) ou "" pour dissocier l'énumération

Description

La commande **OBJET FIXER LISTE PAR NOM** définit, remplace ou dissocie l'*énumération* associée à l'objet ou au groupe d'objets désigné(s) par *objet*. L'énumération dont le nom est passé dans le paramètre *énumération* doit avoir été créée dans l'éditeur d'énumérations, en mode Développement.

Cette commande peut être appliquée, dans un formulaire entrée ou un formulaire de dialogue, aux champs et variables saisissables dont les valeurs peuvent être saisies sous forme de texte.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Note : Cette commande ne peut pas être utilisée avec des champs placés dans le formulaire "liste" d'un sous-formulaire.

La commande **OBJET FIXER LISTE PAR NOM** vous permet de définir ou de remplacer tous les types d'énumérations associées à l'objet ou aux objets désigné(s) par les paramètres *objet* et * : listes de choix (énumérations), listes de valeurs obligatoires et listes de valeurs exclues. Pour cela, il vous suffit de passer dans le paramètre *typeListe* une des constantes suivantes du thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Liste énumération	Entier long	0	Liste simple de choix de valeurs (option "Enumération" dans la Liste des propriétés) (défaut)
Liste exclusions	Entier long	2	Liste de valeurs non acceptées pour la saisie (option "Exclusions" dans la Liste des propriétés)
Liste obligations	Entier long	1	Liste des seules valeurs acceptées pour la saisie (option "Obligations" dans la Liste des propriétés)

Si vous omettez ce paramètre, la valeur 0 (Liste énumération) est utilisée par défaut.

Pour dissocier dans le process courant une liste associée à l'*objet*, il suffit de passer une chaîne vide ("") dans le paramètre *énumération* pour le type de liste concerné.

Exemple 1

L'exemple suivant définit l'énumération liée à un champ Coursiers. Si l'envoi doit être effectué de nuit, alors l'énumération affiche les sociétés de courses qui fonctionnent la nuit. Sinon, l'énumération standard est proposée :

```
Si([Courses]Nuit)
   OBJET FIXER LISTE PAR NOM([Courses]Coursier;"Coursiers de nuit")
Sinon
   OBJET FIXER LISTE PAR NOM([Courses]Coursier;"Coursiers standard")
Fin de si
```

Exemple 2

Associer la liste "choix_coul" en tant qu'énumération simple au pop up/Liste déroulante "CoulPorte" :

```
OBJET FIXER LISTE PAR NOM(*;"CoulPorte";Liste énumération;"choix_coul")
    // dans ce cas le 3e paramètre (constante) peut être omis
```

Exemple 3

Vous souhaitez associer la liste "choix_coul" à une combo box "CoulMur". Comme la combo box est saisissable, vous souhaitez que certaines couleurs telles que "noir", "violet"... ne puissent être utilisées. Ces couleurs sont placées dans la liste "coul_exclues" :

```
OBJET FIXER LISTE PAR NOM(*;"CoulMur";Liste énumération;"choix_coul")
OBJET FIXER LISTE PAR NOM(*;"CoulMur";Liste exclusions;"coul_exclues")
```

Exemple 4

Vous souhaitez supprimer des associations de listes :

```
    // retrait de l'énumération simple
OBJET FIXER LISTE PAR NOM(*;"CoulPorte";Liste énumération;"")
    // retrait de la liste de valeurs non autorisées
OBJET FIXER LISTE PAR NOM(*;"CoulMur";Liste exclusions;"")
```

OBJET FIXER LISTE PAR REFERENCE

OBJET FIXER LISTE PAR REFERENCE ({* ;} objet {; typeListe}; liste)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
typeListe	Entier long	⇒	Type de liste : Liste énumération, Liste obligations ou Liste exclusions
liste	RefListe	⇒	Numéro de référence de liste

Description

La commande **OBJET FIXER LISTE PAR REFERENCE** définit ou remplace l'énumération associée à l'objet ou aux objets désigné(s) par les paramètres *objet* et *, avec la liste hiérarchique référencée dans le paramètre *liste*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Par défaut, si vous omettez le paramètre *typeListe*, la commande définit une énumération source (choix de valeurs) pour l'objet. Le paramètre *typeListe* vous permet de désigner tout type d'énumération. Pour cela, il vous suffit de passer dans ce paramètre une des constantes suivantes du thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Liste énumération	Entier long	0	Liste simple de choix de valeurs (option "Enumération" dans la Liste des propriétés) (défaut)
Liste exclusions	Entier long	2	Liste de valeurs non acceptées pour la saisie (option "Exclusions" dans la Liste des propriétés)
Liste obligations	Entier long	1	Liste des seules valeurs acceptées pour la saisie (option "Obligations" dans la Liste des propriétés)

Passez dans *liste* le numéro de référence de la liste hiérarchique que vous souhaitez associer à l'objet. Cette liste doit avoir été générée à l'aide de la commande **Copier liste**, **Charger liste** ou **Nouvelle liste**.

Pour mettre fin à l'association d'une *liste* à un *objet*, il suffit de passer 0 dans le paramètre *liste* pour le type de liste concerné.

Supprimer une association de liste ne supprime pas la référence de liste en mémoire. N'oubliez pas d'appeler la commande **SUPPRIMER LISTE** lorsque vous n'avez plus besoin d'une liste.

Cette commande est particulièrement intéressante dans le contexte d'un pop up ou d'une combo box associé(e) à une variable ou à un champ (cf. manuel *Mode Développement*). Dans ce cas, l'association est dynamique et toute modification dans la liste est reportée dans le formulaire. Lorsque l'objet est associé à un tableau, la liste est recopiée dans le tableau et les modifications de la liste ne seront pas automatiquement disponibles (cf. exemple 5).

Exemple 1

Association d'une énumération simple (type de liste par défaut) à un champ texte :

```
vListCountries:=Nouvelle liste
AJOUTER A LISTE(vListCountries;"Espagne";1)
AJOUTER A LISTE(vListCountries;"Portugal";2)
AJOUTER A LISTE(vListCountries;"Grèce";3)
OBJET FIXER LISTE PAR REFERENCE([Contact]Country;vListCountries)
```

Exemple 2

Associer la liste "vColor" en tant qu'énumération simple au pop up/Liste déroulante "CoulPorte" :

```
vColor:=Nouvelle liste
AJOUTER A LISTE(vColor;"Bleu";1)
AJOUTER A LISTE(vColor;"Vert";2)
AJOUTER A LISTE(vColor;"Jaune";3)
AJOUTER A LISTE(vColor;"Rose";4)
OBJET FIXER LISTE PAR REFERENCE(*;"CoulPorte";Liste énumération;vColor)
```

Exemple 3

Vous souhaitez maintenant associer la liste "vColor" à une combo box "CoulMur". Comme la combo box est saisissable, vous souhaitez que certaines couleurs telles que "noir", "violet"... ne puissent pas être utilisées. Ces couleurs sont placées dans la liste "vRejet" :

```
OBJET FIXER LISTE PAR REFERENCE(*;"CoulMur";Liste énumération;vColor)
vRejet:=Nouvelle liste
AJOUTER A LISTE(vRejet;"Noir";1)
AJOUTER A LISTE(vRejet;"Gris";2)
AJOUTER A LISTE(vRejet;"Violet";3)
OBJET FIXER LISTE PAR REFERENCE(*;"CoulMur";Liste exclusions;vRejet)
```

Exemple 4

Vous souhaitez supprimer des associations de listes :

```
OBJET FIXER LISTE PAR REFERENCE(*;"CoulMur";Liste énumération;0)
OBJET FIXER LISTE PAR REFERENCE(*;"CoulMur";Liste obligations;0)
OBJET FIXER LISTE PAR REFERENCE(*;"CoulMur";Liste exclusions;0)
```

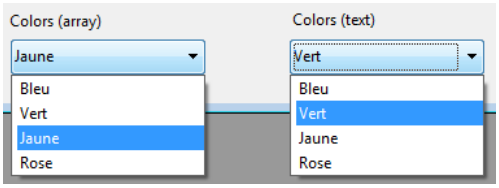
Exemple 5

Cet exemple illustre la différence de fonctionnement de la commande selon qu'elle est appliquée à un pop up menu associé à un tableau texte ou à une variable texte. Dans un formulaire se trouvent deux pop up menus :

Le contenu des pop up menus est défini par la liste *<>vColor* (contenant des valeurs de couleurs). Le code suivant est exécuté au chargement du formulaire :

```
TABLEAU TEXTE(tab1;0) //pop up tab1
C_TEXTE(text1) //pop up text1
OBJET FIXER LISTE PAR REFERENCE(*;"tab1";<>vColor)
OBJET FIXER LISTE PAR REFERENCE(*;"text1";<>vColor)
```

A l'exécution, les deux menus proposent alors les mêmes valeurs :

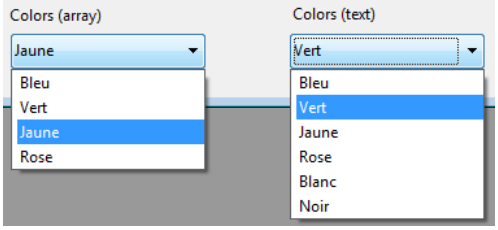


(*Montage montrant simultanément le contenu des menus*)

Vous exécutez alors le code suivant, par exemple via un bouton :

```
AJOUTER A LISTE(<>vColor;"Blanc";5)
AJOUTER A LISTE(<>vColor;"Noir";6)
```

Seul le menu associé au champ texte est mis à jour (via la référence dynamique) :



Pour pouvoir mettre à jour la liste associée au pop up géré par tableau, il est nécessaire de rappeler la commande **OBJET FIXER LISTE PAR REFERENCE** afin de recopier le contenu de la liste.

OBJET FIXER MENU CONTEXTUEL

OBJET FIXER MENU CONTEXTUEL ({* ;} objet ; menuContext)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
menuContext	Booléen	⇒	Vrai = activer menu contextuel, Faux = désactiver menu contextuel

Description

La commande **OBJET FIXER MENU CONTEXTUEL** vous permet d'activer ou de désactiver, pour le process courant, l'association d'un menu contextuel par défaut à l'objet ou aux objets désigné(s) par les paramètres *objet* et *.

L'option "Menu contextuel" est disponible pour les zones de saisie de type texte, les zones Web et les images. Elle permet d'associer à ces objets un menu d'action standard en fonction du type d'objet (par exemple Copier/coller pour les objets texte). Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez **Vrai** dans le paramètre *menuContext* pour activer le menu contextuel, et **Faux** pour le désactiver.

OBJET FIXER MESSAGE AIDE

OBJET FIXER MESSAGE AIDE ({* ;} objet ; messageAide)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
messageAide	Texte	⇒	Contenu du message d'aide

Description

La commande **OBJET FIXER MESSAGE AIDE** permet de définir ou de modifier dynamiquement le message d'aide associé à l'objet ou aux objets désigné(s) par les paramètres *objet* et * pour le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Un message d'aide apparaît à l'exécution du formulaire sous forme d'infobulle à chaque fois que le curseur de la souris survole le champ ou l'objet. Les messages d'aide peuvent également être définis via l'éditeur de formulaires et l'éditeur de structure en mode Développement.

Passez dans le paramètre *messageAide* le contenu du message. Vous pouvez passer soit :

- une chaîne de caractères, par exemple "Utilisez le / comme séparateur",
- une chaîne vide "" pour supprimer l'infobulle.

OBJET FIXER MULTILIGNE

OBJET FIXER MULTILIGNE ({* ;} objet ; multiLigne)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
multiLigne	Entier long	⇒	Statut de la propriété multiligne

Description

La commande **OBJET FIXER MULTILIGNE** vous permet de modifier la propriété "Multilignes" de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

La propriété "Multilignes" permet de contrôler deux paramètres relatifs à l'affichage et à l'impression des zones de texte : l'affichage des mots situés en fin de ligne dans les zones mono-lignes et l'insertion automatique de retours à la ligne. Pour plus d'informations, reportez-vous au paragraphe **Multilignes** du manuel *Mode Développement*. Si vous appliquez cette commande à un objet ne prenant pas en charge cette propriété, la commande ne fait rien.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans le paramètre *multiLigne* la nouvelle valeur de l'option que vous souhaitez définir. Vous pouvez utiliser les constantes suivantes, placées dans le thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Multiligne Auto	Entier long	0	Dans les zones mono-lignes, les mots situés en fin de ligne sont tronqués et il n'y a pas de retours à la ligne. Dans les zones multi-lignes, 4D effectue des retours à la ligne automatiques
Multiligne Non	Entier long	2	Il n'y a aucun retour à la ligne : le texte est toujours affiché sur une seule ligne. Si le champ ou la variable alpha ou texte contient des retour chariots, le texte situé après le premier retour chariot est effacé dès que la zone est modifié
Multiligne Oui	Entier long	1	Dans les zones mono-lignes, le texte est affiché jusqu'au premier retour chariot ou au dernier mot affichable en entier. 4D insère des retours à la ligne, il est possible de faire défiler le contenu de la zone en appuyant sur la touche flèche basse. Dans les zones multi-lignes, 4D effectue des retours à la ligne automatiques

Exemple

Vous souhaitez interdire le multiligne dans une zone de saisie :

```
OBJET FIXER MULTILIGNE(*;"vSaisie";Multiligne Non)
```

OBJET FIXER OPTIONS GLISSER DEPOSER

OBJET FIXER OPTIONS GLISSER DEPOSER ({* ;} objet ; glissable ; glissableAuto ; déposable ; déposableAuto)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
glissable	Booléen	⇒	Glissable = Vrai, sinon Faux
glissableAuto	Booléen	⇒	Glisser automatique = Vrai, sinon Faux
déposable	Booléen	⇒	Déposable = Vrai, sinon Faux
déposableAuto	Booléen	⇒	Déposer automatique = Vrai, sinon Faux

Description

La commande **OBJET FIXER OPTIONS GLISSER DEPOSER** permet de définir ou de modifier dynamiquement les options de glisser-déposer pour l'objet ou les objets désigné(s) par les paramètres *objet* et * pour le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Passez dans chaque paramètre un booléen indiquant si l'option correspondante est active ou inactive :

- *glissable* = Vrai : objet glissable en mode programmé.
- *glissableAuto* = Vrai (utilisable uniquement avec les champs et variables texte, combo box et list box) : objet glissable en mode automatique.
- *déposable* = Vrai : objet acceptant le déposer en mode programmé.
- *déposableAuto* = Vrai (utilisable uniquement avec les champs et variables image, texte, combo box et list box) : objet acceptant le déposer en mode automatique.

Exemple

Définition d'une zone de texte en glisser-déposer auto :

```
OBJET FIXER OPTIONS GLISSER DEPOSER(*;"Comments";Faux;Vrai;Faux;Vrai)
```

OBJET FIXER ORIENTATION TEXTE

OBJET FIXER ORIENTATION TEXTE ({* ;} objet ; orientation)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
orientation	Entier long	⇒	Valeur d'orientation de l'objet

Description

La commande **OBJET FIXER ORIENTATION TEXTE** vous permet de modifier l'orientation du contenu de l'objet ou des objets désigné(s) par les paramètres *objet* et * pour le process courant.

La propriété "Orientation", accessible dans l'éditeur de formulaires, permet d'effectuer des rotations de zones de texte de façon permanente. A la différence de cette propriété, la commande **OBJET FIXER ORIENTATION TEXTE** applique la rotation au contenu de l'objet mais pas à l'objet lui-même. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).
Seuls les textes statiques ainsi que les variables et les champs non saisissables peuvent subir une rotation. Si vous appliquez la commande à un objet ne prenant pas en charge l'orientation de texte, la commande ne fait rien.

Passez dans le paramètre *orientation* l'orientation absolue que vous souhaitez affecter à l'objet. Vous devez passer une des constantes suivantes, placées dans le thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Orientation 0°	Entier long	0	Pas de rotation (valeur par défaut)
Orientation 180°	Entier long	180	Orientation du texte à 180° dans le sens horaire
Orientation 90° droite	Entier long	90	Orientation du texte à 90° dans le sens horaire
Orientation 90° gauche	Entier long	270	Orientation du texte à 90° dans le sens anti-horaire

Note : Seuls les angles correspondant à ces valeurs sont pris en charge. Si vous passez une autre valeur, elle sera ignorée.

Exemple

Vous souhaitez appliquer une orientation de 270° à une variable de votre formulaire :

```
OBJET FIXER SAISISSABLE(*;"maVar";Faux)
  //obligatoire si la variable est saisissable
 OBJET FIXER ORIENTATION TEXTE(*;"maVar";Orientation 90° gauche)
```

OBJET FIXER POLICE

OBJET FIXER POLICE ({* ;} objet ; police)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
police	Chaîne	⇒	Nom de police de caractères

Description

OBJET FIXER POLICE affiche *objet* avec la police définie dans le paramètre *police*. Le paramètre *police* doit contenir un nom de police valide.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Exemple 1

L'exemple suivant définit la police d'un bouton nommé *bOK*. La police est Arial, une police système sous Windows :

```
OBJET FIXER POLICE(bOK;"Arial") ` Modification de la police de MonBouton
```

Exemple 2

L'exemple suivant définit la police de tous les objets d'un formulaire dont le nom contient "info".

```
OBJET FIXER POLICE(*;"@info@";"Times")
```

Exemple 3

L'exemple suivant utilise l'option spéciale *%password*, destinée à la saisie et l'affichage des champs de type “mots de passe”. Lorsque vous passez "%password" dans le paramètre *police* pour un *objet* :

- chaque caractère saisi dans l'objet est représenté par un même symbole,
- les actions "copier" et "couper" sont désactivées dans l'objet.

Note : L'option *%password* est utilisable avec les objets de type champ, variable et combo box.

```
OBJET FIXER POLICE([Utilisateurs]MotPasse;"%password")
```

OBJET FIXER RAYON ARRONDI

OBJET FIXER RAYON ARRONDI ({* ;} objet ; rayon)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
rayon	Entier long	⇒	Nouveau rayon des angles arrondis (en pixels)

Description

La commande **OBJET FIXER RAYON ARRONDI** vous permet de modifier le rayon des angles du ou des objet(s) de type rectangle arrondi dont vous avez passé le nom dans le paramètre *objet*. Le nouveau rayon est défini pour le process uniquement, il n'est pas stocké dans le formulaire.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

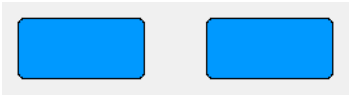
Note : Dans la version actuelle de 4D, cette commande s'applique uniquement aux rectangles arrondis (qui sont des objets statiques). Par conséquent, seule la syntaxe basée sur le nom d'objet utilisant le paramètre * est prise en charge.

Passez dans le paramètre *rayon* la nouvelle valeur de rayon en pixels pour les angles de l'objet. Par défaut, la valeur est de 5 pixels.

Note : Cette valeur peut également être modifiée au niveau du formulaire via la Liste des propriétés (cf. **Rayon d'arrondi (rectangles)**).

Exemple

Votre formulaire contient les rectangles suivants, nommés "Rect1" et "Rect2" :



Vous pouvez exécuter le code suivant afin de changer leurs angles arrondis :

```
OBJET FIXER RAYON ARRONDI(*;"Rect@";20)
```

OBJET FIXER RECTANGLE FOCUS INVISIBLE

OBJET FIXER RECTANGLE FOCUS INVISIBLE ({* ;} objet ; invisible)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
invisible	Booléen	⇒	Vrai = rectangle focus caché, Faux = rectangle focus visible

Description

La commande **OBJET FIXER RECTANGLE FOCUS INVISIBLE** permet de définir ou de modifier dynamiquement l'option d'invisibilité du rectangle de focus de l'objet ou des objets désigné(s) par les paramètres *objet* et * pour le process courant. Ce paramétrage correspond à l'option **Cacher rectangle de focus** disponible pour les objets saisissables dans la Liste des propriétés en mode Développement.

Note : Cette option est utilisable sous Mac OS uniquement. Elle n'a pas d'effet sous Windows.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable ou un champ. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Passez **Vrai** dans le paramètre *invisible* pour cacher le rectangle de focus et **Faux** pour le laisser visible.

OBJET FIXER REDIMENSIONNEMENT

OBJET FIXER REDIMENSIONNEMENT ({* ;} objet ; horizontal ; vertical)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
horizontal	Entier long	⇒	Option de redimensionnement horizontal
vertical	Entier long	⇒	Option de redimensionnement vertical

Description

La commande **OBJET FIXER REDIMENSIONNEMENT** permet de définir ou de modifier dynamiquement les options de redimensionnement de l'objet ou des objets désigné(s) par les paramètres *objet* et * pour le process courant. Ces options définissent l'affichage de l'objet en cas de redimensionnement de la fenêtre du formulaire.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Passez dans le paramètre *horizontal* une valeur indiquant l'option de redimensionnement horizontal que vous souhaitez définir pour l'*objet*. Vous pouvez passer une des constantes suivantes, placées dans le thème **Objets de formulaire (Propriétés)** :

Constante	Type	Valeur	Comment
Redim horizontal agrandir	Entier long	1	Si la fenêtre s'agrandit de 50% en largeur, l'objet s'élargit de 50% vers la droite
Redim horizontal aucun	Entier long	0	Si la fenêtre s'agrandit en largeur, ni la largeur ni la position de l'objet ne varient
Redim horizontal déplacer	Entier long	2	Si la fenêtre s'agrandit de 100 pixels en largeur, l'objet se déplace de 100 pixels vers la droite

Passez dans le paramètre *vertical* une valeur indiquant l'option de redimensionnement vertical que vous souhaitez définir pour l'objet. Vous pouvez passer une des constantes suivantes, placées dans le thème **Objets de formulaire (Propriétés)** :

Constante	Type	Valeur	Comment
Redim vertical agrandir	Entier long	1	Si la fenêtre s'agrandit de 50% en hauteur, l'objet s'allonge de 50% vers le bas
Redim vertical aucun	Entier long	0	Si la fenêtre s'agrandit en hauteur, ni la hauteur ni la position de l'objet ne varient
Redim vertical déplacer	Entier long	2	Si la fenêtre s'agrandit de 100 pixels en hauteur, l'objet se déplace de 100 pixels vers le bas

OBJET FIXER SAISISSABLE

OBJET FIXER SAISISSABLE ({* ;} objet ; saisissable)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une table, un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Table ou Champ ou Variable (si * omis)
saisissable	Booléen	⇒	Vrai = saisissable ; Faux = non saisissable

Description

OBJET FIXER SAISISSABLE rend saisissable ou non saisissable le ou les objet(s) de formulaire désigné(s) par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est une table, un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de table, de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

L'utilisation de cette commande est équivalente à la sélection de l'option Saisissable pour un champ ou une variable dans la Liste des propriétés de l'éditeur de formulaires. **OBJET FIXER SAISISSABLE** fonctionne avec un sous-formulaire uniquement si elle se trouve dans la méthode formulaire du sous-formulaire.

Lorsque *zoneSaisie* est saisissable (Vrai), l'utilisateur peut y placer le curseur pour saisir des données. Lorsque *zoneSaisie* est non saisissable (Faux), l'utilisateur ne peut pas placer le curseur dans la zone et ne peut donc pas saisir de valeurs.

La commande **OBJET FIXER SAISISSABLE** permet également d'activer par programmation le mode "Saisie en liste" pour les sous-formulaires et les formulaires liste affichés par les commandes **MODIFIER SELECTION** et **VISUALISER SELECTION** :

- Pour les sous-formulaires, vous pouvez passer dans le paramètre *objet* soit le nom de la table du sous-formulaire, soit le nom de l'objet sous-formulaire lui-même, par exemple : **OBJET FIXER SAISISSABLE**(*;"Sousform";Vrai)
- Pour les formulaires liste, vous devez passer le nom de la table du formulaire dans le paramètre *objet*, par exemple : **OBJET FIXER SAISISSABLE**([MaTable];Vrai).

Rendre un objet non saisissable n'empêche pas sa modification par programmation.

Note : Vous rendez une cellule de list box non saisissable en passant la valeur -1 à $0 dans l'événement Sur avant saisie, cf. paragraphe **Gestion de la saisie**.

Exemple 1

L'exemple suivant définit un champ de type d'expédition suivant le poids d'un colis expédié. Si le colis pèse un kilo ou moins, l'expéditeur sera La Poste et le champ est rendu non saisissable. Sinon, le champ est rendu saisissable.

```
Si([Expédition]Poids<=1)
   [Expédition]Type:="La Poste"
   OBJET FIXER SAISISSABLE([Expédition]Type;Faux)
Sinon
   OBJET FIXER SAISISSABLE([Expédition]Type;Vrai)
Fin de si
```

Exemple 2

Voici la méthode objet d'une case à cocher placée dans l'en-tête d'une liste pour contrôler le mode Saisie en liste :

```
C_BOOLEEN(bSaisissable)
OBJET FIXER SAISISSABLE([Table1];bSaisissable)
```

OBJET FIXER SOURCE DONNEES

OBJET FIXER SOURCE DONNEES ({* ;} objet ; sourceDonnées)

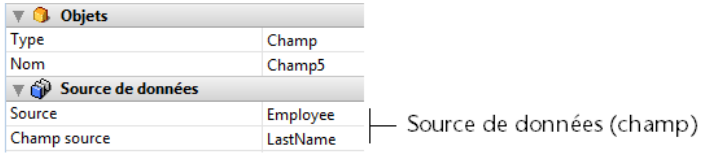
Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
sourceDonnées	Pointeur	⇒	Pointeur vers la nouvelle source de données de l'objet

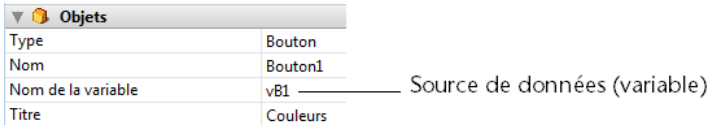
Description

La commande **OBJET FIXER SOURCE DONNEES** vous permet de modifier la source de données de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

La source de données est le champ ou la variable dont la valeur est représentée par l'objet lors de l'exécution du formulaire. En mode Développement, la source de données est définie dans la Liste de propriétés, généralement via les lignes Source et Champ source (champs) ou Nom de la variable (variables) :





Hormis pour les list box (cf. ci-dessous), toutes les sources de données du formulaire peuvent être modifiées par cette commande. Il appartient au développeur de s'assurer de la cohérence des modifications effectuées.

Dans le cas des list box, les points suivants sont à considérer :

- les modifications de sources de données doivent tenir compte du type de list box : par exemple, il n'est pas possible d'utiliser un champ comme source de données de colonne d'une list box de type tableau.
- pour les list box de type sélection, il n'est pas possible de modifier ou de lire la source de données de l'objet list box lui-même : il s'agit dans ce cas d'une référence interne et non d'une source de données.
- cette commande est utile principalement dans le contexte des list box de type tableau. Pour les list box de type sélection, vous pouvez plutôt utiliser la commande **LISTBOX FIXER FORMULE COLONNE**.

Si la commande est appliquée à une source de données non modifiable, elle ne fait rien.

Exemple

Modification de la source de données d'une zone de saisie :

```
C_POINTEUR($ptrChp)
$ptrChp:=Champ(3;2)
OBJET FIXER SOURCE DONNEES(*;"Input";$ptrChp)
```

OBJET FIXER SOUS FORMULAIRE

OBJET FIXER SOUS FORMULAIRE ({* ;} objet {; laTable}; sousFormDetail {; sousFormListe})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
laTable	Table	⇒	Table du formulaire (si formulaire table)
sousFormDetail	Texte	⇒	Nom du formulaire détail du sous-formulaire
sousFormListe	Texte	⇒	Nom du formulaire liste du sous-formulaire (formulaire table)

Description

La commande **OBJET FIXER SOUS FORMULAIRE** vous permet de modifier dynamiquement le formulaire détaillé ainsi que, optionnellement, le formulaire liste écran associé à l'objet sous-formulaire désigné par les paramètres *objet* et *.

Note : Cette commande ne permet pas de changer le type du sous-formulaire lui-même (liste ou page). Cette propriété peut être définie en mode Développement uniquement.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Passez dans le paramètre *laTable* la table des formulaires à utiliser. Ce paramètre est optionnel, vous pouvez l'omettre si vous définissez un formulaire projet comme sous-formulaire détaillé.

Passez dans le paramètre *sousFormDetail* le nom du formulaire à utiliser comme sous-formulaire détaillé.

Passez dans le paramètre *sousFormListe* le nom du formulaire à utiliser comme sous-formulaire en liste. Ce paramètre ne peut être passé que si vous modifiez un sous-formulaire de type liste.

Lorsque vous modifiez un sous-formulaire en page, la commande peut être exécutée à tout moment, les éventuelles sélections courantes ne sont pas modifiées. En revanche, si vous modifiez un sous-formulaire en liste, il ne peut être modifié que lorsqu'il affiche la liste. Si la commande est exécutée alors que le formulaire détaillé est affiché à la suite d'un double-clic dans la liste, une erreur est générée.

OBJET FIXER STYLE BORDURE

OBJET FIXER STYLE BORDURE ({* ;} objet ; styleBordure)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
styleBordure	Entier long	⇒	Style de la ligne de bordure

Description

La commande **OBJET FIXER STYLE BORDURE** vous permet de modifier le style de la ligne de bordure de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

La propriété "Style de bordure" permet de modifier l'apparence du contour des objets. Pour plus d'informations, reportez-vous au paragraphe **Style de la bordure** dans le manuel *Mode Développement*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans le paramètre *styleBordure* la valeur de style de ligne que vous souhaitez appliquer à l'objet. Vous pouvez passer une des constantes suivantes, placées dans le thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Bordure Aucune	Entier long	0	Les objets apparaissent sans encadrement
Bordure Double	Entier long	5	Les objets apparaissent encadrés par une double ligne, c'est-à-dire deux lignes continues de 1 pt séparées par un pixel
Bordure Normal	Entier long	1	Les objets apparaissent dans un cadre continu de 1 pt
Bordure Relief	Entier long	3	Les objets apparaissent avec un effet 3D (relief)
Bordure Relief inversé	Entier long	4	Les objets apparaissent avec un effet 3D en creux (relief inversé)
Bordure Système	Entier long	6	Le cadre est dessiné en fonction des spécifications graphiques du système
Bordure Trait pointillé	Entier long	2	Les objets apparaissent dans un cadre pointillé de 1 pt

OBJET FIXER STYLE POLICE

OBJET FIXER STYLE POLICE ({* ;} objet ; style)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
style	Entier long	⇒	Style de police

Description

OBJET FIXER STYLE POLICE assigne le style de police *style* à ou aux objet(s) de formulaire désigné(s) par *objet*.
Le nombre *style* est un code de style du système d'exploitation. En additionnant des codes, vous combinez les styles.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Vous devez passer dans le paramètre *style* une des constantes prédéfinies suivantes ou la somme de plusieurs de ces constantes (thème "**Styles de caractères**") :

Constante	Type	Valeur
Gras	Entier long	1
Italique	Entier long	2
Normal	Entier long	0
Souligné	Entier long	4

Exemple 1

L'exemple suivant définit le style de police du bouton *bAjoutNouveau*. Le style demandé est gras italique :

```
OBJET FIXER STYLE POLICE(bAjoutNouveau;Gras+Italique)
```

Exemple 2

L'exemple suivant définit le style de police Normal pour tous les objets de formulaire dont le nom débute par "vt" :

```
OBJET FIXER STYLE POLICE(*;"vt@";Normal)
```

OBJET FIXER TAILLE POLICE

OBJET FIXER TAILLE POLICE ({* ;} objet ; taille)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
taille	Entier long	⇒	Taille de police en points

Description

OBJET FIXER TAILLE POLICE définit la taille de la police du ou des objet(s) de formulaire spécifié(s) par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

La *taille* peut être tout Entier compris entre 1 et 255. Si la taille exacte n'existe pas, les caractères sont proportionnellement redimensionnés.

La zone de l'objet, telle qu'elle a été définie dans le formulaire, doit être suffisamment grande pour afficher les données dans la nouvelle taille. Autrement, le texte peut être tronqué ou pas du tout affiché.

Exemple 1

L'exemple suivant définit la taille de police de la variable appelée *vInfo* :

```
OBJET FIXER TAILLE POLICE(vInfo;14)
```

Exemple 2

L'exemple suivant définit la taille de police de tous les objets de formulaire dont le nom débute par "hl" :

```
OBJET FIXER TAILLE POLICE(*;"hl@";14)
```

OBJET FIXER TEXTE EXEMPLE

OBJET FIXER TEXTE EXEMPLE ({* ;} objet ; texteExemple)

Paramètre	Type		Description
*	Opérateur	⇨	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇨	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
texteExemple	Texte	⇨	Texte d'exemple associé à l'objet

Description

La commande **OBJET FIXER TEXTE EXEMPLE** vous permet d'associer un texte d'exemple à l'objet ou aux objets désigné(s) par les paramètres *objet* et *.

Pour plus d'informations sur les textes d'exemple, reportez-vous au manuel *Mode Développement*.

Si un texte d'exemple avait déjà été associé à l'objet via la Liste des propriétés, il est remplacé dans le process courant par le contenu du paramètre *texteExemple*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans *texteExemple* le texte d'aide ou l'indication devant apparaître lorsque l'objet est vide.

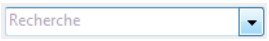
Note : L'insertion de références xliff dans les textes d'exemple n'est pas prise en charge par la commande **OBJET FIXER TEXTE EXEMPLE**. Cette possibilité n'existe que pour les textes d'exemple définis via la Liste des propriétés.

Cette commande peut être utilisée uniquement avec les objets de formulaire de type variable, champ et combo box. Un texte d'exemple peut être associé à des valeurs de type texte et alpha. Il peut également être associé à des données de type date ou heure si l'objet de formulaire comporte la propriété "Vide si null".

Exemple

Vous souhaitez afficher le texte exemple "Recherche" dans une combo box :

```
OBJET FIXER TEXTE EXEMPLE(*;"comb_rech";"Recherche")
```



OBJET FIXER TITRE

OBJET FIXER TITRE ({* ;} objet ; libellé)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
libellé	Chaîne	⇒	Nouveau libellé de l'objet

Description

La commande **OBJET FIXER TITRE** change le libellé du ou des objets(s) spécifié(s) dans le paramètre *objet* et le remplace par la valeur définie dans le paramètre *libellé.*

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable (variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

OBJET FIXER TITRE peut s'appliquer à tous les types d'objets simples contenant un libellé :

- boutons et boutons 3D,
- cases à cocher et cases à cocher 3D,
- boutons radio et boutons radio 3D,
- en-têtes de list box,
- textes statiques,
- zones de groupe.

Généralement, cette commande s'applique à un objet à la fois. La zone de libellé de l'objet doit être assez grande pour pouvoir accueillir le texte ; sinon, le texte est tronqué.

N'utilisez pas de retours chariot dans *libellé*.

Si vous souhaitez définir un libellé sur plusieurs lignes, utilisez le caractère "\" ("\\" dans l'éditeur de code) comme retour à la ligne. Cette possibilité est permise pour les boutons 3D, cases à cocher 3D, boutons radio 3D et les en-têtes de list box.

Note : Passez "\\\" si vous souhaitez utiliser le caractère "\" dans le libellé.

Exemple 1

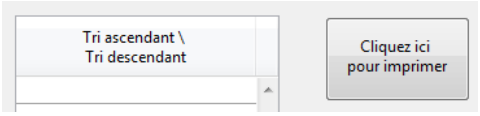
L'exemple suivant est la méthode objet d'un bouton de recherche situé dans la zone de pied de page d'un formulaire sortie affiché par la commande **MODIFIER SELECTION**. La méthode effectue une recherche dans une table et active ou inactive le bouton intitulé *bSuppr* et change son titre, en fonction des résultats de la recherche :

```
CHERCHER([Personnes];[Personnes]Nom=vNom)
Au cas ou
   :(Enregistrements trouves([Personnes])=0) // Personne n'a été trouvé
      OBJET FIXER TITRE(bSuppr;" Supprimer")
      OBJET FIXER ACTIVATION(bSuppr;Faux)
   :(Enregistrements trouves([Personnes])=1) // Une personne a été trouvée
      OBJET FIXER TITRE(bSuppr;"Supprimer la personne")
      OBJET FIXER ACTIVATION(bSuppr;Vrai)
   :(Enregistrements trouves([Personnes])>1) // Plusieurs personnes ont été trouvées
      OBJET FIXER TITRE(bSuppr;"Supprimer les personnes")
      OBJET FIXER ACTIVATION(bSuppr;Vrai)
Fin de cas
```

Exemple 2

Vous souhaitez insérer des libellés sur deux lignes :

```
OBJET FIXER TITRE(*;"entete1";"Tri ascendant \\\ \\Tri descendant")
OBJET FIXER TITRE(*;"bouton1";"Cliquez ici \\pour imprimer")
```



OBJET FIXER TYPE INDICATEUR

OBJET FIXER TYPE INDICATEUR ({* ;} objet ; indicateur)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
indicateur	Entier long	⇒	Type d'indicateur

Description

La commande **OBJET FIXER TYPE INDICATEUR** vous permet de modifier le type d'indicateur de progression du ou des thermomètre(s) désigné(s) par les paramètres *objet* et * pour le process courant.

Le type d'indicateur définit la variante d'affichage du thermomètre. Pour plus d'informations, reportez-vous à la section **Jauges** dans le manuel *Mode Développement*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans le paramètre *indicateur* le type d'indicateur à afficher. Vous pouvez utiliser une des constantes suivantes, placées dans le thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Indicateur Barber shop	Entier long	2	Barre affichant une animation continue
Indicateur Barre de progression	Entier long	1	Barre de progression standard
Indicateur de progression asynchrone	Entier long	3	Indicateur circulaire affichant une animation continue

OBJET FIXER VALEUR MAXIMUM

OBJET FIXER VALEUR MAXIMUM ({* ;} objet ; valeurMaxi)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
valeurMaxi	Date, Heure, Numérique	⇒	Valeur maximale pour l'objet

Description

La commande **OBJET FIXER VALEUR MAXIMUM** permet de modifier la valeur maximum de l'objet ou des objets désigné(s) par les paramètres *objet* et * pour le process courant.

La propriété "Valeur maximum" peut être appliquée aux données de type numérique, date ou heure. Pour plus d'informations, reportez-vous au paragraphe **Valeurs maximales et minimales** dans le manuel *Mode Développement*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans *valeurMaxi* la nouvelle valeur maximum à affecter à l'*objet* pour le process courant. Cette valeur doit correspondre au type de l'objet, sinon l'erreur 18 "Les types sont incompatibles" est retournée.

OBJET FIXER VALEUR MINIMUM

OBJET FIXER VALEUR MINIMUM ({* ;} objet ; valeurMini)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
valeurMini	Date, Heure, Numérique	⇒	Valeur minimale pour l'objet

Description

La commande **OBJET FIXER VALEUR MINIMUM** permet de modifier la valeur minimum de l'objet ou des objets désigné(s) par les paramètres *objet* et * pour le process courant.

La propriété "Valeur minimum" peut être appliquée aux données de type numérique, date ou heure. Pour plus d'informations, reportez-vous au paragraphe **Valeurs maximales et minimales** dans le manuel *Mode Développement*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans *valeurMini* la nouvelle valeur minimum à affecter à l'objet pour le process courant. Cette valeur doit correspondre au type de l'objet, sinon l'erreur 18 "Les types sont incompatibles" est retournée.

OBJET FIXER VISIBLE

OBJET FIXER VISIBLE ({* ;} objet ; visible)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est passé) ou Champ ou Variable (si * est omis)
visible	Booléen	⇒	Vrai = visible, Faux = invisible

Description

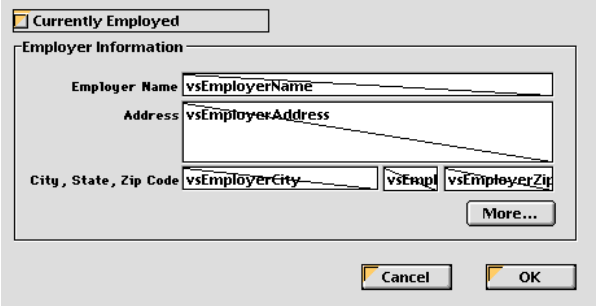
La commande **OBJET FIXER VISIBLE** affiche ou masque le ou les objet(s) défini(s) par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre *objet* désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables objets uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section **Objets de formulaires**.

Si vous passez la valeur **VRAI** dans le paramètre *visible*, le ou les objet(s) sont affichés. Si vous passez **FAUX** dans *visible*, les objets sont masqués.

Exemple

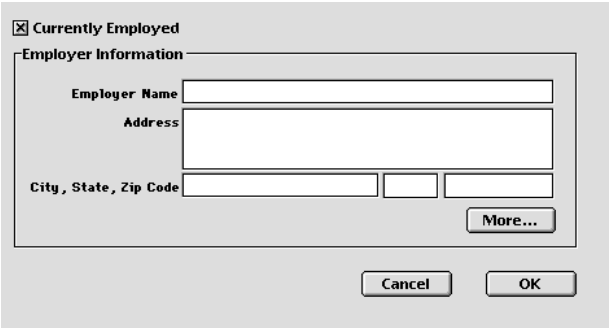
Voici un formulaire tel qu'il apparaît en mode Développement :



Les objets dans la zone de groupe **Employer Information** ont tous un nom qui contient l'expression “employer” (y compris la zone de groupe). Lorsque l'option **Currently Employed** est cochée, les objets doivent être visibles, lorsqu'elle est désélectionnée les objets doivent être invisibles. Voici la méthode projet de la case à cocher :

```
   ` Méthode objet Case à cocher cbCurrentlyEmployed
Au cas ou
   :(Evenement formulaire=Sur chargement)
      cbCurrentlyEmployed:=1

   :(Evenement formulaire=Sur clic)
   ` Cacher ou montrer tous les objets dont le nom contient "emp"
      OBJET FIXER VISIBLE(*;"@emp@";cbCurrentlyEmployed # 0)
   ` Mais toujours conserver la case à cocher visible
      OBJET FIXER VISIBLE(cbCurrentlyEmployed;Vrai)
Fin de cas
```

En exécution, le formulaire apparaîtra ainsi :



ou ainsi :


Currently Employed
Cancel
OK

OBJET Lire action

OBJET Lire action ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Texte	⇐	Action standard associée

Description

La commande **OBJET Lire action** retourne le code de l'action standard associée à l'objet ou aux objets désigné(s) par les paramètres *objet* et *.

L'action standard d'un objet peut avoir été définie en mode Développement via la Liste des propriétés (cf. section **Actions standard** dans le manuel *Mode Développement*) ou à l'aide de la commande **OBJET FIXER ACTION**.

La commande retourne une chaîne contenant le code de l'action associée à l'objet. Vous pouvez comparer la valeur reçue aux constantes du thème "**Valeurs Texte pour Action standard associée**" :

Constante	Type	Valeur
Objet action Actualiser URL courant	Chaîne	39
Objet action Afficher Presse papiers	Chaîne	23
Objet action Ajouter sous enreg	Chaîne	14
Objet action Aller à page	Chaîne	15
Objet action Annuler	Chaîne	1
Objet action Annuler édition	Chaîne	17
Objet action Arrêter chargement URL	Chaîne	40
Objet action Coller	Chaîne	20
Objet action Copier	Chaîne	19
Objet action Couper	Chaîne	18
Objet action CSM	Chaîne	36
Objet action Dernier enreg	Chaîne	6
Objet action Dernière page	Chaîne	11
Objet action Effacer	Chaîne	21
Objet action Enreg précédent	Chaîne	4
Objet action Enreg suivant	Chaîne	3
Objet action Modifier sous enreg	Chaîne	12
Objet action Ouvrir URL précédent	Chaîne	37
Objet action Ouvrir URL suivant	Chaîne	38
Objet action Page précédente	Chaîne	9
Objet action Page suivante	Chaîne	8
Objet action Premier enreg	Chaîne	5
Objet action Première page	Chaîne	10
Objet action Propriétés de la base	Chaîne	32
Objet action Quitter	Chaîne	27
Objet action Répéter	Chaîne	31
Objet action Retour au mode Développement	Chaîne	35
Objet action Séparateur auto	Chaîne	16
Objet action Supprimer enreg	Chaîne	7
Objet action Supprimer sous enreg	Chaîne	13
Objet action Test application	Chaîne	26
Objet action Tout sélectionner	Chaîne	22
Objet action Valider	Chaîne	2
Objet sans action standard	Chaîne	0

Exemple

Vous souhaitez associer l'action "Annuler" à tous les objets sans action du formulaire :

```
TABLEAU TEXTE($tabObj;0)

FORM LIRE OBJETS($tabObj)
Boucle($i;1;Taille tableau($tabObj))
   Si(OBJET Lire action(*;$tabObj{$i})=Objet sans action standard)
      OBJET FIXER ACTION(*;$tabObj{$i};Objet action Annuler)
   Fin de si
Fin de boucle
```

OBJET Lire activation

OBJET Lire activation ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Variable (si * omis)
Résultat	Booléen	⇐	Vrai = objet(s) activé(s), Faux sinon

Description

La commande **OBJET Lire activation** retourne Vrai si l'objet ou le groupe d'objets désigné par *objet* est activé dans le formulaire et Faux s'il est inactivé.

Un objet activé réagit aux clics souris et aux raccourcis clavier.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable (variable objet uniquement).

Cette commande peut être appliquée aux types d'objets suivants :

- Bouton, Bouton par défaut, Bouton 3D, Bouton invisible, Bouton inversé
- Bouton radio, Bouton radio 3D, Bouton image
- Case à cocher, Case à cocher 3D
- Pop-up menu, Liste déroulante, Combo Box, Menu/Liste déroulante
- Thermomètre, Règle

OBJET Lire alignement horizontal

OBJET Lire alignement horizontal ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est passé) ou Champ ou variable (si * est omis)
Résultat	Entier long	⇐	Code d'alignement

Description

La commande **OBJET Lire alignement horizontal** retourne un code indiquant le type d'alignement horizontal appliqué à l'objet désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre *objet* désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables de formulaire uniquement).

Note : Si vous appliquez la commande à un ensemble d'objets, seule la valeur d'alignement du dernier objet est retournée.

Le code retourné correspond à l'une des constantes suivantes, placées dans le thème **Objets de formulaire (Propriétés)** :

Constante	Type	Valeur	Comment
Aligné à droite	Entier long	4	
Aligné à gauche	Entier long	2	
Aligné au centre	Entier long	3	
Aligné par défaut	Entier long	1	
wk justify	Entier long	5	Alignement justifié. Disponible uniquement pour les zones 4D Write Pro.

Note : La constante wk justify est disponible dans le thème "**4D Write Pro**".

Les objets de formulaire auxquels un alignement peut être appliqué sont les suivants :

- Zones de défilement
- Combo box
- Textes statiques
- Zones de groupes
- Pop up menus/Listes déroulantes
- Champs
- Variables
- List box
- Colonnes de list box
- En-têtes de list box
- Pieds de list box
- Zones **4D Write Pro**

OBJET Lire alignement vertical

OBJET Lire alignement vertical ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	⇐	Type d'alignement

Description

La commande **OBJET Lire alignement vertical** retourne une valeur indiquant le type d'alignement vertical appliqué à l'objet désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Note : Si vous appliquez la commande à un ensemble d'objets, seule la valeur d'alignement du dernier objet est retournée.

La valeur retournée correspond à l'une des constantes suivantes du thème **Objets de formulaire (Propriétés)** :

Constante	Type	Valeur	Comment
Aligné au centre	Entier long	3	
Aligné en bas	Entier long	4	
Aligné en haut	Entier long	2	

Les objets de formulaire auxquels un alignement vertical peut être appliqué sont les suivants :

- list box,
- colonnes de list box,
- en-tête et pieds de list box.

OBJET LIRE BARRES DEFILEMENT

OBJET LIRE BARRES DEFILEMENT ({* ;} objet ; horizontale ; verticale)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Variable ou champ (si * omis)
horizontale	Booléen, Entier long	⇐	Visibilité de la barre horizontale
verticale	Booléen, Entier long	⇐	Visibilité de la barre verticale

Description

La commande **OBJET LIRE BARRES DEFILEMENT** permet de connaître le statut affiché/masqué des barres de défilement horizontale et verticale de l'objet ou du groupe d'objets désigné par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Vous pouvez passer dans *horizontale* et *verticale* des variables de type booléen ou entier long :

- si vous passez des variables booléennes, la valeur retournée reflètera le statut **courant** de la barre de défilement :
  - si la barre de défilement a été définie comme masquée, le paramètre reçoit Faux,
  - si la barre de défilement a été définie comme affichée, le paramètre reçoit Vrai,
  - si la barre de défilement a été définie en mode automatique, le paramètre reçoit Vrai ou Faux en fonction de l'affichage courant de l'objet.
- si vous passez des variables entier long, la valeur retournée reflètera la visibilité définie pour la barre de défilement :
  - si la barre de défilement a été définie comme masquée, le paramètre reçoit 0,
  - si la barre de défilement a été définie comme affichée, le paramètre reçoit 1,
  - si la barre de défilement a été définie en mode automatique, le paramètre reçoit 2.

Cette commande est utilisable avec les objets de formulaire suivants :

- Champs et variables objet texte ou image,
- List box,
- Listes hiérarchiques,
- Sous-formulaires.

Pour plus d'informations, reportez-vous à la description de la commande **OBJET FIXER BARRES DEFILEMENT**.

OBJET Lire case a cocher trois etats

OBJET Lire case a cocher trois etats ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Booléen	⇐	Vrai = case à cocher à trois états, Faux = case à cocher standard

Description

La commande **OBJET Lire case a cocher trois etats** retourne l'état courant de la propriété "Trois états" de la ou des case(s) à cocher désignée(s) par les paramètres *objet* et *.

La propriété "Trois états" peut avoir été définie soit via la Liste des propriétés, soit via la commande **OBJET FIXER CASE A COCHER TROIS ETATS** si elle a été appelée dans le process courant.

OBJET Lire configuration clavier

OBJET Lire configuration clavier ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
Résultat	Chaîne	↩	Code de la langue de configuration , "" = pas de configuration

Description

La commande **OBJET Lire configuration clavier** retourne la configuration clavier courante associée à l'objet ou aux objets désigné(s) par les paramètres *objet* et * pour le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable ou un champ. Dans ce cas, vous ne passez pas une nom mais une référence.

La commande retourne une chaîne indiquant le code de langue utilisé, basé sur les RFC3066, ISO639 et ISO3166. Pour plus d'informations, reportez-vous à la description de la commande **FIXER LANGUE BASE**.

OBJET LIRE COORDONNEES

OBJET LIRE COORDONNEES ({* ;} objet ; gauche ; haut ; droite ; bas)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
gauche	Entier long	⇐	Coordonnée gauche de l'objet
haut	Entier long	⇐	Coordonnée supérieure de l'objet
droite	Entier long	⇐	Coordonnée droite de l'objet
bas	Entier long	⇐	Coordonnée inférieure de l'objet

Description

La commande **OBJET LIRE COORDONNEES** retourne dans les variables ou champs *gauche*, *haut*, *droite* et *bas* les coordonnées (en points) du ou des objet(s) du formulaire courant défini(s) par les paramètres * et *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne de caractères). Si vous ne passez pas le paramètre *, vous indiquez que *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable de type objet uniquement).

Si vous passez un nom d'objet dans le paramètre *objet* et utilisez le caractère joker @ afin de sélectionner plusieurs objets, les coordonnées retournées seront celles du rectangle formé par l'ensemble des objets concernés.

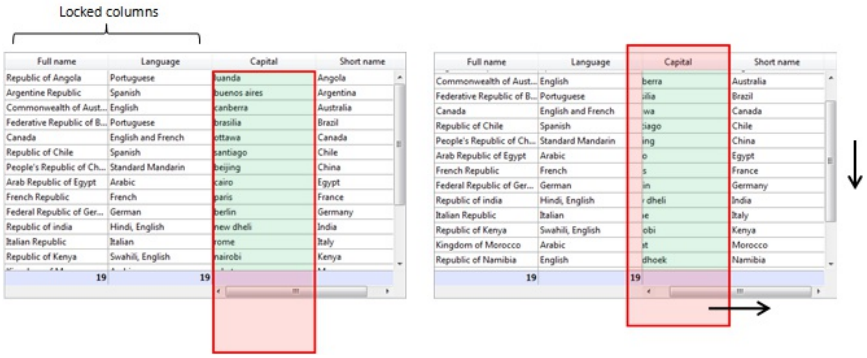
Note : Il est possible de paramétrer le mode d'interprétation du caractère @, lorsque celui-ci est inclus dans une chaîne de caractères. Cette option influe sur le fonctionnement des commandes du thème "Propriétés des objets". Pour plus d'informations, reportez-vous au manuel Mode Développement.

Si l'objet n'existe pas ou si la commande est appelée ailleurs que dans le contexte d'un formulaire, les coordonnées retournées sont (0;0;0;0).

Dans le contexte des list box, la commande **OBJET LIRE COORDONNEES** peut retourner les coordonnées de parties spécifiques des list box, c.-à-d. des colonnes, en-têtes ou pieds, ou celles de l'objet list box parent. Dans les versions de 4D antérieures à la v14 R5, cette commande retournait toujours les coordonnées de la list box parente, quelle que soit la zone passée en paramètre. Désormais, lorsque le paramètre *objet* référence un en-tête, une colonne ou un pied de list box, ce sont les coordonnées de ce sous-objet qui sont retournées. Vous pouvez utiliser ce fonctionnement, par exemple, pour afficher une petite icône dans la cellule d'en-tête d'une list box lorsqu'elle est survolée par le curseur, indiquant à l'utilisateur qu'il peut cliquer pour afficher un menu contextuel.
Pour des raisons de cohérence, le cadre de référence utilisé par la commande est le même, qu'elle travaille avec un objet list box ou un sous-objet de la list box : le point d'origine est le coin supérieur gauche du formulaire contenant l'objet. Pour les sous-objets de list box, les coordonnées retournées sont théoriques ; elles prennent en compte le défilement appliqué à la list box avant tout éventuel *clipping* (c.-à-d. le découpage effectué en fonction des coordonnées de la list box parente). Par conséquent, le sous-objet peut ne pas être visible, ou être partiellement visible, et ses coordonnées peuvent se trouver en-dehors des limites du formulaire (voire être négatives). Pour déterminer si le sous-objet est visible (et quelle partie est visible), vous devez comparer les coordonnées retournées avec celles de la list box elle-même, en tenant compte des règles suivantes :

- Les limites des sous-objets dépendent des coordonnées de leur list box parente (telles que retournées par la commande **OBJET LIRE COORDONNEES** pour la list box).
- Les sous-objets en-tête et pied sont affichés au-dessus du contenu de la colonne : lorsque les coordonnées d'une colonne coupent celles d'une ligne d'en-tête ou de pied, la colonne n'est pas affichée à l'emplacement de l'intersection.
- Les éléments des colonnes verrouillées sont affichés au-dessus des éléments des colonnes défilables : lorsque les coordonnées d'un élément d'une colonne défilable croisent celles d'un élement d'une colonne verrouillée, il n'est pas affiché à l'emplacement de l'intersection.

Par exemple, examinez le schéma suivant, dans lequel les coordonnées de la colonne *Capital* sont symbolisées par un rectangle rouge :



Comme vous pouvez le voir dans la première image, la colonne est plus grande que la list box, donc ses coordonnées dépassent la limite basse de la list box, pied inclus. Dans la seconde image, la list box a défilé, et donc la colonne a également été déplacée "sous" les zones de la colonne *Language* et d'en-tête. Dans tous les cas, pour calculer la partie réellement visible de la colonne (représentée par la zone verte), vous devez soustraire les zones rouges.

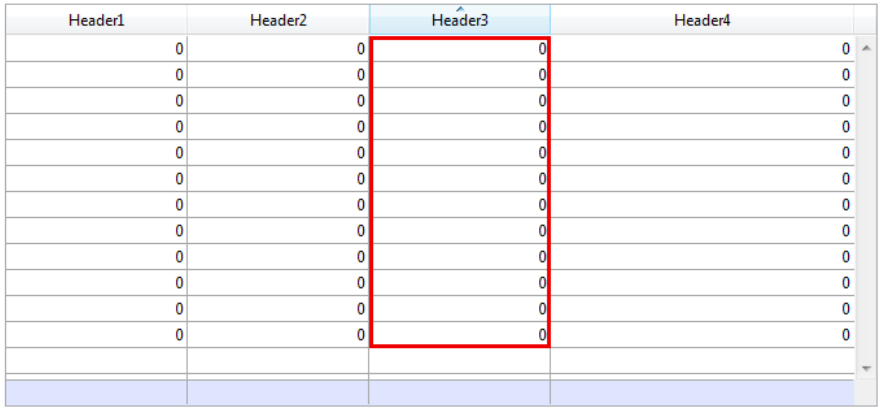
Exemple 1

Vous souhaitez obtenir les coordonnées du rectangle formé par tous les objets dont le nom commence par "bouton" :

```
OBJET LIRE COORDONNEES(*;"bouton@";gauche;haut;droite;bas)
```

Exemple 2

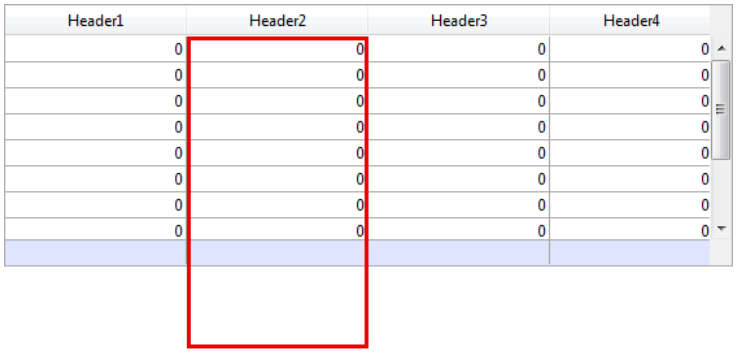
Pour les besoins de votre interface, vous souhaitez entourer d'un rectangle rouge la zone sur laquelle l'utilisateur a cliqué :



Dans la méthode objet de la list box, vous écrivez :

```
OBJET FIXER VISIBLE(*;"rectangleInfo";Faux) //initialiser un rectangle rouge
$ptr:=OBJET Lire pointeur(Objet courant)
OBJET LIRE COORDONNEES($ptr->;$x1;$y1;$x2;$y2)
OBJET FIXER VISIBLE(*;"RedRect";Vrai)
OBJET FIXER COORDONNEES(*;"RedRect";$x1;$y1;$x2;$y2)

OBJET LIRE COORDONNEES(*;"LB1";$lbx1;$lby1;$lbx2;$lby2)
Si($lby1>$y1)|($lby2<$y2) // si la zone cliquée est en-dehors de la list box
   OBJET FIXER VISIBLE(*;"Alerte";Vrai) //afficher une alerte
Sinon
   OBJET FIXER VISIBLE(*;"Alerte";Faux)
Fin de si
```

La méthode retourne les coordonnées théoriques. Si la list box est redimensionnée, vous pourrez avoir besoin de calculer le détourage afin de savoir quelle partie est visible :



OBJET Lire correction orthographique

OBJET Lire correction orthographique ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
Résultat	Booléen	⇐	Vrai = correction automatique, Faux = pas de correction automatique

Description

La commande **OBJET Lire correction orthographique** retourne le statut de l'option **Correction orthographique** du ou des objet(s) désigné(s) par les paramètres *objet* et * pour le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable ou un champ. Dans ce cas, vous ne passez pas une nom mais une référence.

La commande retourne **Vrai** si la correction automatique est activée pour l'*objet*, et **Faux** sinon.

OBJET LIRE COULEURS RVB

OBJET LIRE COULEURS RVB ({* ;} objet ; couleurAvantPlan {; couleurArrièrePlan {; couleurArrièrePlanAlt}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
couleurAvantPlan	Entier long	⇐	Valeur de la couleur RVB d'avant-plan
couleurArrièrePlan	Entier long	⇐	Valeur de la couleur RVB d'arrière-plan
couleurArrièrePlanAlt	Entier long	⇐	Valeur de la couleur RVB d'arrière-plan alternée

Description

La commande **OBJET LIRE COULEURS RVB** retourne les couleurs d'avant-plan et d'arrière-plan de l'objet ou du groupe d'objets désigné(s) par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Lorsque la commande est appliquée à un objet de type List box, la couleur d'arrière plan alternée des lignes peut être retournée dans le paramètre *couleurArrièrePlanAlt*. Dans ce cas, la valeur de *couleurArrièrePlan* est utilisée pour le fond des lignes impaires uniquement.

Les valeurs de couleurs RVB retournées dans les paramètres *couleurAvantPlan*, *couleurArrièrePlan* et *couleurArrièrePlanAlt* peuvent être soit des entiers longs de 4 octets au format (0x00RRGGBB), soit des valeurs négatives correspondant aux couleurs "système". Dans ce dernier cas, vous pouvez comparer la valeur obtenue aux constantes du thème **FIXER COULEUR RVB** :

Constante	Type	Valeur	Comment
Coul arrière plan	Entier long	-2	
Coul claire	Entier long	-4	
Coul de fond texte sélect	Entier long	-7	
Coul fond élément sélect désact	Entier long	-11	
Coul fond ligne menu sélect	Entier long	-9	
Coul fond transparent	Entier long	-16	Cette constante peut être utilisée uniquement avec les paramètres *couleurArrièrePlan* et *couleurArrièrePlanAlt*.
Coul premier plan	Entier long	-1	
Coul sombre	Entier long	-3	
Coul texte ligne menu sélect	Entier long	-10	
Coul texte sélect	Entier long	-8	

Note : Les couleurs "système" sont appliquées à l'aide de de la commande **OBJET FIXER COULEURS RVB**.

Pour plus d'informations sur le format des paramètres *couleurAvantPlan*, *couleurArrièrePlan* et *couleurArrièrePlanAlt*, reportez-vous à la description de la commande **OBJET FIXER COULEURS RVB**.

OBJET LIRE DEFILEMENT

OBJET LIRE DEFILEMENT ({* ;} objet ; positionLigne {; positionH})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
positionLigne	Entier long	⇐	Numéro de la première ligne affichée ou Défilement vertical en pixels (images)
positionH	Entier long	⇐	Numéro de la première colonne affichée (list box) ou Défilement horizontal en pixels (images)

Description

La commande **OBJET LIRE DEFILEMENT** retourne dans les paramètres *positionLigne* et *positionH* des informations relatives à la position des barres de défilement de l'objet de formulaire désigné par les paramètres * et *objet*.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *objet* est le nom d'un objet de type sous-formulaire, liste hiérarchique, zone de défilement, list box ou image (dans ce cas, passez une chaîne dans objet). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable (*RefListe* de liste hiérarchique, image ou list box) ou un champ.

Note : Avec les objets de type sous-formulaire, seule la syntaxe avec * est prise en charge.

Si *objet* désigne un objet de type liste (sous-formulaire, liste hiérarchique, zone de défilement ou list box), *positionLigne* retourne le numéro de la première ligne affichée dans l'objet. *positionH* (list box uniquement) retourne le numéro de la première colonne affichée dans la partie gauche de la list box. Avec les autres types d'objets, ce paramètre retourne 0.

Si *objet* désigne une image (variable ou champ), *positionLigne* retourne le décalage vertical et *positionH* le décalage horizontal de l'image. Ces valeurs sont exprimées en pixels par rapport à l'origine de l'image dans le système de coordonnées locales.

OBJET LIRE EQUIVALENT CLAVIER

OBJET LIRE EQUIVALENT CLAVIER ({* ;} objet ; touche ; modifiers)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
touche	Chaîne	⇐	Touche associée à l'objet
modifiers	Entier long	⇐	Masque ou combinaison de masques de touche(s) de modification

Description

La commande **OBJET LIRE EQUIVALENT CLAVIER** retourne l'équivalent clavier associé à l'objet ou aux objets désigné(s) par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Le paramètre *touche* retourne le caractère associé à la touche (dans le cas d'une touche standard) ou une chaîne entre crochets désignant la touche (dans le cas d'une touche de fonction). Vous pouvez comparer cette valeur aux constantes du thème **Touches équivalents clavier** (cf. commande **OBJET FIXER EQUIVALENT CLAVIER**).

Le paramètre *modifiers* retourne une valeur indiquant la ou les touche(s) de modification associée(s) à l'équivalent clavier. Si plusieurs touches de modification ont été combinées, la commande retourne le cumul de leurs valeurs. Vous pouvez comparer la valeur reçue aux constantes suivantes du thème **Evénements (Modifiers)** :

Constante	Type	Valeur	Comment
Masque touche commande	Entier long	256	Touche Ctrl sous Windows, touche Commande sous OS X
Masque touche contrôle	Entier long	4096	Touche Ctrl sous OS X, ou clic droit sous Windows et OS X
Masque touche majuscule	Entier long	512	Windows et OS X
Masque touche option	Entier long	2048	Touche Alt (aussi appelée Option sous OS X)

Si aucune touche de modification n'a été définie dans l'équivalent clavier, *modifiers* retourne 0.

Note : Si le paramètre *objet* désigne plusieurs objets du formulaire ayant des paramétrages différents, la commande retourne "" dans *touche* et 0 dans *modifiers*.

OBJET LIRE EVENEMENTS

OBJET LIRE EVENEMENTS ({* ;} objet ; tabEvénements)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet ou "" pour désigner le formulaire (si * est spécifié) ou Champ ou variable (si * est omis)
tabEvénements	Tableau entier long	⇐	Tableau des événements activés

Description

La commande **OBJET LIRE EVENEMENTS** vous permet de d'obtenir la configuration courante des événements formulaire du formulaire, de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

Les événements formulaire peuvent avoir été activés/désactivés soit via la Liste des propriétés, soit via la commande **OBJET FIXER EVENEMENTS** si elle a été appelée dans le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).
Pour obtenir la configuration des événements du formulaire lui-même, passez le paramètre optionnel * et une chaîne vide "" dans *objet* : dans ce cas, vous désignez le formulaire courant.

Note : Si vous souhaitez obtenir les événements d'un sous-formulaire lié à une table, seule la syntaxe basée sur le nom d'objet peut être utilisée.

Passez dans le paramètre *tabEvénements* un tableau Entier long. A l'exécution de la commande, ce tableau est automatiquement dimensionné et reçoit tous les événements formulaire prédéfinis ou personnalisés activés pour l'objet ou le formulaire. Vous pouvez comparer les valeurs reçues aux constantes du thème "**Evénements formulaire**".

Attention, le tableau *tabEvénements* est retourné vide si aucune méthode objet n'est associée à l'*objet* ou si aucune méthode formulaire n'est associée au formulaire.

Exemple

Vous souhaitez activer deux événements et obtenir la liste des événements pour un objet :

```
TABLEAU ENTIER LONG($TabCurEvents;0)
TABLEAU ENTIER LONG($TabActiv;2)
$TabActiv{1}:=Sur clic entête
$TabActiv{2}:=Sur clic pied
OBJET FIXER EVENEMENTS(*;"Col1";$TabActiv;Activer événements autres inchangés)
OBJET LIRE EVENEMENTS(*;"Col1";$TabCurEvents)
```

OBJET Lire feuille de style

OBJET Lire feuille de style ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Texte	⇐	Nom de la feuille de style

Description

La commande **OBJET Lire feuille de style** retourne le nom de la feuille de style associée à l'objet ou aux objets désigné(s) par les paramètres *objet* et *.

La feuille de style peut avoir été affectée en mode Développement via la Liste des propriétés ou pour le process courant via la commande **OBJET FIXER FEUILLE DE STYLE**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

La commande peut retourner soit :

- un nom de feuille de style,
- une chaîne vide ("") si aucune feuille de style n'est affectée,
- une des valeurs de constantes suivantes du thème "**Styles de caractères**" si une feuille de style automatique est affectée :

Constante	Type	Valeur	Comment
Feuille de style automatique	Chaîne	__automatic__	Utilisée par défaut pour tous les objets
Feuille de style automatique_additionnel	Chaîne	__automatic_additional_text__	Prise en charge par les textes statiques, champs et variables uniquement. Utilisée pour du texte additionnel dans les boîtes de dialogue.
Feuille de style automatique_texte principal	Chaîne	__automatic_main_text__	Prise en charge par les textes statiques, champs et variables uniquement. Utilisée pour le texte principal des boîtes de dialogue.

Si la commande désigne plusieurs objets, la feuille de style retournée n'est significative que si elle est affectée à tous les objets.

OBJET Lire filtre saisie

OBJET Lire filtre saisie ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * spécifié) ou Variable ou champ (si * omis)
Résultat	Texte	⇐	Nom du filtre de saisie

Description

La commande **OBJET Lire filtre saisie** retourne le nom du filtre de saisie éventuellement associé à l'objet ou au groupe d'objets désigné par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

OBJET Lire formatage

OBJET Lire formatage ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	➡	Nom d'objet (si * spécifié) ou Champ ou variable (si * omis)
Résultat	Chaîne	⮌	Format d'affichage de l'objet

Description

La commande **OBJET Lire formatage** retourne le format d'affichage courant appliqué à l'objet spécifié par le paramètre *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (dans ce cas, passez une chaîne dans *objet*). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable.

Cette commande retourne le format d'affichage courant de l'objet, c'est-à-dire le format défini en mode Développement ou à l'aide de la commande **OBJET FIXER FORMATAGE**. **OBJET Lire formatage** fonctionne avec tous les types d'objets de formulaire (champs ou variables) acceptant un format d'affichage : booléen, date, heure, image, chaîne, numérique, ainsi que les grilles de boutons, cadrans, thermomètres, règles, pop up menus image, boutons image, boutons 3D et en-têtes de list box. Pour plus d'informations sur les formats d'affichage de ces objets, reportez-vous à la documentation de la commande **OBJET FIXER FORMATAGE**.

Note : Si vous appliquez la commande à un ensemble d'objets, seul le formatage du dernier objet pris en compte est retourné.

Lorsque la commande **OBJET Lire formatage** est appliquée à des objets de type date, heure ou image (formats définis sous forme de constantes), la chaîne retournée correspond au code de caractere de la constante. Pour obtenir la valeur de la constante, il suffit d'appliquer la fonction **Code de caractere** au résultat (cf. exemple ci-dessous).

Exemple 1

Cet exemple permet d'obtenir la valeur de la constante de formatage appliquée à la variable image dont le nom d'objet est "maphoto" :

```
C_ALPHA(2;$format)
OBJET FIXER FORMATAGE(*;"maphoto";Caractere(Sur fond))
 `Application du format sur fond (valeur = 3)
$format:=OBJET Lire formatage(*;"maphoto")
ALERTE("Format numéro :"+Chaine(Code de caractere($format)))
 `Affichage de la valeur "3"
```

Exemple 2

Cet exemple permet d'obtenir le formatage appliqué au champ booléen [Adhérents]Etat_civil :

```
C_ALPHA(30;$format)
$format:=OBJET Lire formatage([Adhérents]Etat_civil)
ALERTE($format) `Affichage du format, par exemple "Marié;Célibataire"
```

OBJET LIRE IMPRESSION TAILLE VARIABLE

OBJET LIRE IMPRESSION TAILLE VARIABLE ({* ;} objet ; tailleVariable {; fixeSousForm})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
tailleVariable	Booléen	⇐	Vrai = Impression taille variable, Faux = Impression taille fixe
fixeSousForm	Entier long	⇐	Option d'impression en taille fixe des sous-formulaires

Description

La commande **OBJET LIRE IMPRESSION TAILLE VARIABLE** vous permet de d'obtenir la configuration courante des options d'impression en taille variable de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

Les propriétés d'impression en taille variable peuvent être définies via la Liste des propriétés ou la commande **OBJET FIXER IMPRESSION TAILLE VARIABLE**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

La commande retourne dans le paramètre *tailleVariable* une variable booléenne dont la valeur indique le statut activé (**Vrai**) ou inactivé (**Faux**) de l'impression en taille variable.

Si l'*objet* est un sous-formulaire et si l'impression en taille variable est à **Faux**, la commande retourne également dans le paramètre *fixeSousForm* l'option d'impression en taille fixe du sous-formulaire. Vous pouvez comparer la valeur retournée aux constantes suivantes, placées dans le thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Impression limitée avec report	Entier long	2	La taille initiale de la zone du sous-formulaire est conservée mais 4D imprime le formulaire plusieurs fois pour imprimer tous les enregistrements.
Impression limitée par le cadre	Entier long	1	4D n'imprime que les enregistrements qui apparaissent dans la zone du sous-formulaire. Le formulaire n'est imprimé qu'une fois et les enregistrements qui ne sont pas imprimés sont ignorés.

OBJET Lire liste reference

OBJET Lire liste reference ({* ;} objet {; typeListe}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
typeListe	Entier long	⇒	Type de liste : Liste énumération, Liste obligations ou Liste exclusions
Résultat	RefListe	⇐	Numéro de référence de la liste

Description

La commande **OBJET Lire liste reference** retourne le numéro de référence (*RefListe*) de la liste hiérarchique associée à l'objet ou au groupe d'objets désigné par *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Par défaut, si vous omettez le paramètre *typeListe*, la commande retourne le nom de l'énumération simple (liste de valeurs) associée à l'*objet*. Vous pouvez également obtenir le numéro de référence des listes d'obligations ou d'exclusions en passant dans *typeListe* une des constantes suivantes du thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Liste énumération	Entier long	0	Liste simple de choix de valeurs (option "Enumération" dans la Liste des propriétés) (défaut)
Liste exclusions	Entier long	2	Liste de valeurs non acceptées pour la saisie (option "Exclusions" dans la Liste des propriétés)
Liste obligations	Entier long	1	Liste des seules valeurs acceptées pour la saisie (option "Obligations" dans la Liste des propriétés)

Si aucune liste hiérarchique n'est associée à l'objet pour le *typeListe* défini, la commande retourne 0.

OBJET Lire menu contextuel

OBJET Lire menu contextuel ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Booléen	⇐	Vrai = menu contextuel activé, Faux = menu contextuel désactivé

Description

La commande **OBJET Lire menu contextuel** retourne le statut courant de l'option "Menu contextuel" de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

L'option "Menu contextuel" d'un objet peut avoir été définie en mode Développement via la Liste des propriétés, ou à l'aide de la commande **OBJET FIXER MENU CONTEXTUEL**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

La commande retourne **Vrai** si le menu contextuel est activé pour l'objet et **Faux** dans le cas contraire.

OBJET Lire message aide

OBJET Lire message aide ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Texte	⇐	Message d'aide de l'objet

Description

La commande **OBJET Lire message aide** retourne le message d'aide associé à l'objet ou aux objets désigné(s) par les paramètres *objet* et * dans le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

La commande retourne le message d'aide courant associé à l'objet, qu'il ait été défini en mode Développement ou pour le process à l'aide de la commande **OBJET FIXER MESSAGE AIDE**. La chaîne retournée représente le message tel qu'il apparaît lors de l'exécution du formulaire. S'il contient des éléments variables (*resname* xliff ou références 4D), ils sont interprétés en fonction du contexte.

Exemple

Le libellé d'un bouton image est stocké sous forme de message d'aide. Ce libellé est stocké dans un fichier xliff, il diffère en fonction de la langue courante de l'application :

```
OBJET FIXER MESSAGE AIDE(*;"bouton1";":xliff:btn_discover")
$helpmess:=OBJET Lire message aide(*;"bouton1")
    // $helpmess contient par exemple "Découvrir" avec un 4D français et "Discover" avec un 4D anglais
```

OBJET Lire multiligne

OBJET Lire multiligne ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Entier long	⇐	Statut multiligne de l'objet

Description

La commande **OBJET Lire multiligne** retourne le statut courant de l'option "Multilignes" de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

L'option "Multilignes" d'un objet peut avoir été définie en mode Développement via la Liste des propriétés, ou à l'aide de la commande **OBJET FIXER MULTILIGNE**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

La valeur retournée correspond à l'une des constantes suivantes du thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Multiligne Auto	Entier long	0	Dans les zones mono-lignes, les mots situés en fin de ligne sont tronqués et il n'y a pas de retours à la ligne. Dans les zones multi-lignes, 4D effectue des retours à la ligne automatiques
Multiligne Non	Entier long	2	Il n'y a aucun retour à la ligne : le texte est toujours affiché sur une seule ligne. Si le champ ou la variable alpha ou texte contient des retour chariots, le texte situé après le premier retour chariot est effacé dès que la zone est modifié
Multiligne Oui	Entier long	1	Dans les zones mono-lignes, le texte est affiché jusqu'au premier retour chariot ou au dernier mot affichable en entier. 4D insère des retours à la ligne, il est possible de faire défiler le contenu de la zone en appuyant sur la touche flèche basse. Dans les zones multi-lignes, 4D effectue des retours à la ligne automatiques

Note : Si vous appliquez la commande **OBJET Lire multiligne** à un objet ne prenant pas en charge l'option "Multilignes", la valeur 0 est retournée.

OBJET Lire nom

OBJET Lire nom {(sélecteur)} -> Résultat

Paramètre	Type		Description
sélecteur	Entier long	⇨	Catégorie d'objet
Résultat	Texte	⇦	Nom de l'objet

Description

La commande **OBJET Lire nom** retourne le nom d'un objet de formulaire.

La commande permet de désigner deux types d'objets en fonction du paramètre *sélecteur*. Vous pouvez passer dans ce paramètre l'une des constantes suivantes (placées dans le thème **Objets de formulaire (Accès)**) :

- Objet courant ou *sélecteur* omis : Si vous passez ce sélecteur ou omettez le paramètre *sélecteur*, la commande retourne le nom de l'objet à partir duquel elle a été appelée (méthode objet ou sous-méthode appelée par la méthode objet). Dans ce cas, la commande doit être appelée dans le contexte d'un objet de formulaire, sinon elle retourne une chaîne vide.
- Objet avec focus : Si vous passez ce sélecteur, la commande retourne le nom de l'objet ayant le focus dans le formulaire.

Exemple

Méthode objet du bouton "bValiderForm" :

```
$nomBtn:=OBJET Lire nom(Objet courant)
```

Après l'exécution de cette méthode objet, la variable *$nomBtn* contient la valeur "bValiderForm".

OBJET Lire nom liste

OBJET Lire nom liste ({* ;} objet {; typeListe}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
typeListe	Entier long	⇒	Type de liste : Liste énumération, Liste obligations ou Liste exclusions
Résultat	Texte	⇐	Nom de l'énumération (définie en mode Développement)

Description

La commande **OBJET Lire nom liste** retourne le nom de l'énumération associée à l'objet ou au groupe d'objets désigné par *objet*. 4D vous permet d'associer une énumération (créée avec l'éditeur d'énumérations en mode Développement) aux objets de formulaire via l'éditeur de formulaires ou la commande **OBJET FIXER LISTE PAR NOM**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Le paramètre optionnel *typeListe* vous permet de désigner le type de liste que vous souhaitez obtenir. Par défaut, si vous omettez ce paramètre, la commande retourne le nom de l'énumération simple (liste de valeurs) associée à l'objet. Vous pouvez également obtenir le nom des listes d'obligations ou d'exclusions en passant dans *typeListe* une des constantes suivantes du thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Liste énumération	Entier long	0	Liste simple de choix de valeurs (option "Enumération" dans la Liste des propriétés) (défaut)
Liste exclusions	Entier long	2	Liste de valeurs non acceptées pour la saisie (option "Exclusions" dans la Liste des propriétés)
Liste obligations	Entier long	1	Liste des seules valeurs acceptées pour la saisie (option "Obligations" dans la Liste des propriétés)

Si aucune liste du type défini n'est associée à l'*objet*, la commande retourne une chaîne vide ("").

OBJET LIRE OPTIONS GLISSER DEPOSER

OBJET LIRE OPTIONS GLISSER DEPOSER ({* ;} objet ; glissable ; glissableAuto ; déposable ; déposableAuto)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
glissable	Booléen	⇐	Glissable = Vrai, sinon Faux
glissableAuto	Booléen	⇐	Glisser automatique = Vrai, sinon Faux
déposable	Booléen	⇐	Déposable = Vrai, sinon Faux
déposableAuto	Booléen	⇐	Déposer automatique = Vrai, sinon Faux

Description

La commande **OBJET LIRE OPTIONS GLISSER DEPOSER** retourne les options de glisser-déposer pour l'objet ou les objets désigné(s) par les paramètres *objet* et * pour le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

La commande retourne les options de glisser-déposer courantes, qu'elles aient été définies en mode Développement ou pour le process à l'aide de la commande **OBJET FIXER OPTIONS GLISSER DEPOSER**.

Chaque paramètre retourne Vrai ou Faux suivant que l'option correspondante est active ou inactive :

- *glissable* = Vrai : objet glissable en mode programmé.
- *glissableAuto* = Vrai (utilisable uniquement avec les champs et variables texte, combo box et list box) : objet glissable en mode automatique.
- *déposable* = Vrai : objet acceptant le déposer en mode programmé.
- *déposableAuto* = Vrai (utilisable uniquement avec les champs et variables image, texte, combo box et list box) : objet acceptant le déposer en mode automatique.

OBJET Lire orientation texte

OBJET Lire orientation texte ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Entier long	⇐	Angle de rotation du texte

Description

La commande **OBJET Lire orientation texte** retourne la valeur d'orientation courante appliquée au texte de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

L'option "Orientation" d'un objet peut avoir été définie en mode Développement via la Liste des propriétés ou à l'aide de la commande **OBJET FIXER ORIENTATION TEXTE**.

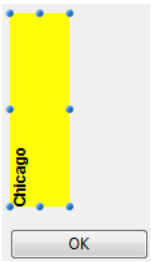
Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

La valeur retournée correspond à l'une des constantes suivantes du thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Orientation 0°	Entier long	0	Pas de rotation (valeur par défaut)
Orientation 180°	Entier long	180	Orientation du texte à 180° dans le sens horaire
Orientation 90° droite	Entier long	90	Orientation du texte à 90° dans le sens horaire
Orientation 90° gauche	Entier long	270	Orientation du texte à 90° dans le sens anti-horaire

Exemple

Soit l'objet suivant (une orientation "90° gauche" lui a été appliquée dans l'éditeur de formulaires) :



Si, à l'exécution du formulaire, vous appelez l'instruction suivante :

```
OBJET FIXER ORIENTATION TEXTE(*;"monTexte";Orientation 180°)
```

... l'objet prend alors l'apparence suivante :


```
$vOrt:=OBJET Lire orientation texte(*;"monTexte") //$vOrt=180
```

OBJET Lire pointeur

OBJET Lire pointeur ({sélecteur }{;}{ nomObjet {; nomSousFormulaire}}) -> Résultat

Paramètre	Type		Description
sélecteur	Entier long	⇒	Catégorie d'objet
nomObjet	Texte	⇒	Nom d'objet
nomSousFormulaire	Texte	⇒	Nom d'objet sous-formulaire
Résultat	Pointeur	⇐	Pointeur sur la variable de l'objet

Description

La commande **OBJET Lire pointeur** retourne un pointeur vers la variable d'un objet de formulaire.

Cette commande permet de désigner différents types d'objets en fonction du paramètre *sélecteur*. Vous pouvez passer dans ce paramètre l'une des constantes suivantes (placées dans le thème **Objets de formulaire (Accès)**) :

- Objet courant ou *sélecteur* omis : Si vous omettez le paramètre *sélecteur* ou passez ce sélecteur, la commande retourne un pointeur vers la variable associée à l'objet courant (objet dont la méthode est en cours d'exécution).
  Note : Ce fonctionnement équivaut strictement à celui de la commande historique **Self**. La commande **Self** est conservée pour des raisons de compatibilité uniquement.
- Objet avec focus : Si vous passez ce sélecteur, la commande retourne un pointeur vers la variable associée à l'objet ayant le focus dans le formulaire. Les deux derniers paramètres optionnels sont ignorés s'ils sont passés.
  Note : Ce fonctionnement équivaut strictement à celui de la commande **Objet focus**. La commande **Objet focus** est obsolète à compter de 4D v12.
- Objet conteneur sous formulaire : Si vous passez ce sélecteur, la commande retourne un pointeur vers la variable liée au conteneur du sous-formulaire. Les deux derniers paramètres optionnels sont ignorés s'ils sont passés. Ce sélecteur ne peut donc être utilisé que dans le contexte d'un formulaire utilisé comme sous-formulaire, afin d'accéder à la variable liée à l'objet conteneur.
- Objet nommé : Si vous passez ce sélecteur, vous devez également passer le deuxième paramètre, *nomObjet*. Dans ce cas, la commande retourne un pointeur vers la variable associée à l'objet dont vous avez passé le nom dans ce paramètre.
  Note : Si *nomObjet* correspond à un sous-formulaire et que l'option "Sous-formulaire liste" est cochée, la commande retourne un pointeur vers la table du sous-formulaire si une table source est définie et Nil dans le cas contraire.

Le paramètre optionnel *nomSousFormulaire* vous permet de récupérer un pointeur vers un objet *nomObjet* n'appartenant pas au contexte courant, c'est-à-dire au formulaire parent. Pour pouvoir utiliser ce paramètre, vous devez avoir passé le sélecteur Objet nommé.
Lorsque le paramètre *nomSousFormulaire* est passé, la commande **OBJET Lire pointeur** recherche dans un premier temps l'objet sous-formulaire nommé *nomSousFormulaire* dans le formulaire courant, puis recherche à l'intérieur de ce sous-formulaire un objet nommé *nomObjet*. Si cet objet est trouvé, elle retourne un pointeur vers la variable de cet objet.

Exemple

Soit un formulaire "SF" utilisé deux fois comme sous-formulaire dans le même formulaire parent. Les objets sous-formulaires sont nommés "SF1" et "SF2". Le formulaire "SF" contient un objet nommé *ValeurCourante*. Dans l'événement "Sur chargement" de la méthode formulaire du formulaire parent, nous souhaitons initialiser l'objet *ValeurCourante* de SF1 à "Janvier" et celui de SF2 "Février" :

```
C_POINTEUR($Ptr)
$Ptr:=OBJET Lire pointeur(Objet nommé;"ValeurCourante";"SF1")
$Ptr->:="Janvier"
$Ptr:=OBJET Lire pointeur(Objet nommé;"ValeurCourante";"SF2")
$Ptr->:="Février"
```

OBJET Lire police

OBJET Lire police ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
Résultat	Texte	⇐	Nom de la police

Description

La commande **OBJET Lire police** retourne le nom de la police de caractères utilisée par le ou les objet(s) de formulaire désigné(s) par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

OBJET Lire rayon arrondi

OBJET Lire rayon arrondi ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Entier long	⮌	Rayon des angles arrondis (en pixels)

Description

La commande **OBJET Lire rayon arrondi** retourne la valeur courante du rayon des angles du ou des objet(s) de type rectangle arrondi dont vous avez passé le nom dans le paramètre *objet*. Cette valeur peut avoir été définie au niveau du formulaire dans la Liste des propriétés (cf. **Rayon d'arrondi (rectangles)**), ou via la commande **OBJET FIXER RAYON ARRONDI** pour le process courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Note : Dans la version actuelle de 4D, cette commande s'applique uniquement aux rectangles arrondis (qui sont des objets statiques). Par conséquent, seule la syntaxe basée sur le nom d'objet utilisant le paramètre * est prise en charge.

Cette commande retourne le rayon des angles arrondis en pixels. Par défaut, la valeur de 5 pixels.

Exemple

Le code suivant peut être associé à la méthode d'un bouton :

```
C_ENTIER LONG($radius)
$radius:=OBJET Lire rayon arrondi(*;"GreenRect") //lire la valeur courante
OBJET FIXER RAYON ARRONDI(*;"GreenRect";$radius+1) //augmenter le rayon
 // La valeur maximale est gérée automatiquement : lorsqu'elle est atteinte,
 // le bouton n'a plus d'effet
```

OBJET Lire rectangle focus invisible

OBJET Lire rectangle focus invisible ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	⇐	Vrai = rectangle focus caché, Faux = rectangle focus visible

Description

La commande **OBJET Lire rectangle focus invisible** retourne le statut de l'option d'invisibilité du rectangle de focus de l'objet ou des objets désigné(s) par les paramètres *objet* et * pour le process courant. Ce paramétrage correspond à l'option **Cacher rectangle de focus** disponible pour les objets saisissables dans la Liste des propriétés en mode Développement. La commande retourne le statut courant de l'option, qu'elle ait été définie en mode Développement ou à l'aide de la commande **OBJET FIXER RECTANGLE FOCUS INVISIBLE**.

Note : Cette option est utilisable sous Mac OS uniquement. Elle n'a pas d'effet sous Windows.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable ou un champ. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

La commande retourne **Vrai** si le rectangle de focus est masqué et **Faux** s'il est visible.

OBJET LIRE REDIMENSIONNEMENT

OBJET LIRE REDIMENSIONNEMENT ({* ;} objet ; horizontal ; vertical)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
horizontal	Entier long	⇐	Option de redimensionnement horizontal
vertical	Entier long	⇐	Option de redimensionnement vertical

Description

La commande **OBJET LIRE REDIMENSIONNEMENT** retourne les options de redimensionnement courantes de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

La commande retourne les options de redimensionnement courantes, qu'elles aient été définies en mode Développement ou pour le process à l'aide de la commande **OBJET FIXER REDIMENSIONNEMENT**. Ces options définissent l'affichage de l'objet en cas de redimensionnement de la fenêtre du formulaire.

Le paramètre *horizontal* retourne une valeur indiquant l'option de redimensionnement horizontal définie pour l'objet. Vous pouvez comparer la valeur reçue aux constantes suivantes, placées dans le thème **Objets de formulaire (Propriétés)** :

Constante	Type	Valeur	Comment
Redim horizontal agrandir	Entier long	1	Si la fenêtre s'agrandit de 50% en largeur, l'objet s'élargit de 50% vers la droite
Redim horizontal aucun	Entier long	0	Si la fenêtre s'agrandit en largeur, ni la largeur ni la position de l'objet ne varient
Redim horizontal déplacer	Entier long	2	Si la fenêtre s'agrandit de 100 pixels en largeur, l'objet se déplace de 100 pixels vers la droite

Le paramètre *vertical* retourne une valeur indiquant l'option de redimensionnement vertical définie pour l'objet. Vous pouvez comparer la valeur reçue aux constantes suivantes, placées dans le thème **Objets de formulaire (Propriétés)** :

Constante	Type	Valeur	Comment
Redim vertical agrandir	Entier long	1	Si la fenêtre s'agrandit de 50% en hauteur, l'objet s'allonge de 50% vers le bas
Redim vertical aucun	Entier long	0	Si la fenêtre s'agrandit en hauteur, ni la hauteur ni la position de l'objet ne varient
Redim vertical déplacer	Entier long	2	Si la fenêtre s'agrandit de 100 pixels en hauteur, l'objet se déplace de 100 pixels vers le bas

OBJET Lire saisissable

OBJET Lire saisissable ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
Résultat	Booléen	⇐	Vrai = objet(s) saisissable(s), Faux sinon

Description

La commande **OBJET Lire saisissable** retourne Vrai si l'objet ou le groupe d'objets désigné par *objet* dispose de l'attribut **saisissable** et Faux sinon.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

OBJET Lire source donnees

OBJET Lire source donnees ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Pointeur	⇐	Pointeur vers la source de données courante de l'objet

Description

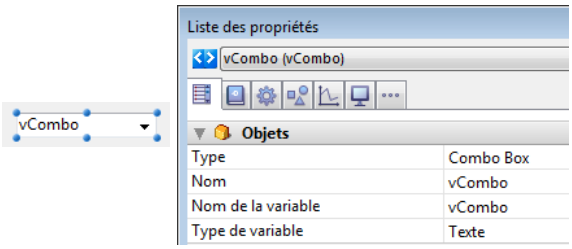
La commande **OBJET Lire source donnees** retourne la source de données courante de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

La source de données d'un objet peut avoir été définie en mode Développement via la Liste des propriétés ou à l'aide de la commande **OBJET FIXER SOURCE DONNEES**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Exemple

Soit un objet combo box défini dans un formulaire :



Vous exécutez le code suivant :

```
$vPtr :=OBJET Lire source donnees(*;"vCombo")
    //$vPtr contient ->vCombo
```

OBJET LIRE SOUS FORMULAIRE

OBJET LIRE SOUS FORMULAIRE ({* ;} objet ; ptrTable ; sousFormDetail {; sousFormListe})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ptrTable	Table	⇐	Pointeur vers la table du formulaire
sousFormDetail	Texte	⇐	Nom du formulaire détail du sous-formulaire
sousFormListe	Texte	⇐	Nom du formulaire liste du sous-formulaire (formulaire table)

Description

La commande **OBJET LIRE SOUS FORMULAIRE** vous permet d'obtenir les noms du ou des formulaire(s) associé(s) à l'objet sous-formulaire désigné par les paramètres *objet* et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

La commande retourne dans le paramètre *ptrTable* un pointeur vers la table du ou des formulaire(s) utilisé(s). Si le sous-formulaire utilise un formulaire projet, le paramètre contient **Nil**.
La commande retourne dans le paramètre *sousFormDetail* le nom du formulaire détaillé utilisé dans le sous-formulaire.
La commande retourne dans le paramètre *sousFormListe* le nom du formulaire liste utilisé dans le sous-formulaire. S'il n'y a pas de formulaire liste, une chaîne vide est retournée.

OBJET Lire style bordure

OBJET Lire style bordure ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Entier long	⇐	Style de la ligne de bordure

Description

La commande **OBJET Lire style bordure** retourne le style de ligne de bordure de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

Le style de bordure d'un objet peut avoir été défini en mode Développement via la Liste des propriétés, ou à l'aide de la commande **OBJET FIXER STYLE BORDURE**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

La commande retourne une valeur correspondant au style de la bordure. Vous pouvez comparer la valeur reçue aux constantes suivantes, placées dans le thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Bordure Aucune	Entier long	0	Les objets apparaissent sans encadrement
Bordure Double	Entier long	5	Les objets apparaissent encadrés par une double ligne, c'est-à-dire deux lignes continues de 1 pt séparées par un pixel
Bordure Normal	Entier long	1	Les objets apparaissent dans un cadre continu de 1 pt
Bordure Relief	Entier long	3	Les objets apparaissent avec un effet 3D (relief)
Bordure Relief inversé	Entier long	4	Les objets apparaissent avec un effet 3D en creux (relief inversé)
Bordure Système	Entier long	6	Le cadre est dessiné en fonction des spécifications graphiques du système
Bordure Trait pointillé	Entier long	2	Les objets apparaissent dans un cadre pointillé de 1 pt

OBJET Lire style police

OBJET Lire style police (* ; objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
Résultat	Entier long	⇐	Style de police

Description

La commande **OBJET Lire style police** retourne le style courant de la police de caractères utilisée par le ou les objet(s) de formulaire désigné(s) par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Vous pouvez comparer la valeur retournée par la commande à la valeur d'une ou plusieurs des constantes prédéfinies suivantes, placées dans le thème **Styles de caractères** :

Constante	Type	Valeur
Normal	Entier long	0
Gras	Entier long	1
Italique	Entier long	2
Souligné	Entier long	4

OBJET LIRE TAILLE OPTIMALE

OBJET LIRE TAILLE OPTIMALE ({* ;} objet ; largeurOpti ; hauteurOpti {; largeurMaxi})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis
largeurOpti	Entier long	⇐	Largeur optimale de l'objet
hauteurOpti	Entier long	⇐	Hauteur optimale de l'objet
largeurMaxi	Entier long	⇒	Largeur maximum de l'objet

Description

La commande **OBJET LIRE TAILLE OPTIMALE** retourne dans les paramètres *largeurOpti* et *hauteurOpti* la largeur et la hauteur "optimales" de l'objet de formulaire désigné par les paramètres * et *objet*. Ces valeurs sont exprimées en pixels. Cette commande est particulièrement utile dans le cadre de l'affichage ou de l'impression d'états complexes, associée à la commande **OBJET DEPLACER**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne de caractères). Si vous ne passez pas le paramètre *, vous indiquez que *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (de type objet uniquement).

Les valeurs optimales retournées indiquent la taille minimale de l'objet pour que son contenu courant soit entièrement inclus dans ses limites. En général, ces valeurs n'ont de sens qu'avec des objets contenant du texte. Ce calcul tient compte de la police, de sa taille, de son style et du contenu de l'objet. Il tient compte également des césures et des retours chariot. A noter que dans le cas des boutons 3D, la commande fonctionne même si le bouton contient uniquement une icône.
Si l'objet spécifié est vide, la *largeurOpti* retournée est 0.

La taille retournée ne tient pas compte du cadre graphique éventuellement appliqué autour de l'objet ni des barres de défilement. Pour obtenir la taille réelle d'un objet à l'écran, il sera nécessaire d'ajouter l'épaisseur de ces éléments.

Le paramètre optionnel *largeurMaxi* vous permet d'attribuer une largeur maximale à l'objet. Si la largeur optimale de l'objet est supérieure à cette valeur, **OBJET LIRE TAILLE OPTIMALE** retourne *largeurMaxi* dans le paramètre *largeurOpti* et augmente la hauteur optimale en conséquence.

Les objets pris en charge par cette commande sont les suivants :

- Zones de texte statiques
- Textes insérés sous forme de références
- Champs et variables de type Alpha, Texte, Réel, Entier, Entier long, Date, Heure, Booléens (cases à cocher et boutons radio)
- Boutons
- Colonnes de list box en contexte d'affichage (seules les lignes visibles sont prises en compte)

Pour tous les autres types d'objets de formulaires (zones de groupes, onglets, rectangles, droites, cercles/ellipses, zones externes, etc.), la commande **OBJET LIRE TAILLE OPTIMALE** retourne la taille courante de l'objet (définie dans l'éditeur de formulaires et éventuellement à l'aide de la commande **OBJET DEPLACER**).

Exemple

Reportez-vous à l'exemple de la routine **FIXER TAQUET IMPRESSION**.

OBJET Lire taille police

OBJET Lire taille police ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
Résultat	Entier long	⇐	Taille de la police en points

Description

La commande **OBJET Lire taille police** retourne la taille (en points) de la police de caractères utilisée par le ou les objet(s) de formulaire désigné(s) par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

OBJET LIRE TAILLE SOUS FORMULAIRE

OBJET LIRE TAILLE SOUS FORMULAIRE (largeur ; hauteur)

Paramètre	Type		Description
largeur	Entier long	⇦	Largeur de l'objet sous-formulaire
hauteur	Entier long	⇦	Hauteur de l'objet sous-formulaire

Description

La commande **OBJET LIRE TAILLE SOUS FORMULAIRE** retourne la *largeur* et la *hauteur* (en pixels) d'un objet sous-formulaire "courant", affiché dans le formulaire parent.

Cette commande doit être appelée depuis la méthode d'un formulaire utilisé en sous-formulaire et affiché dans un objet sous-formulaire. Elle retourne la *largeur* et la *hauteur* de l'objet contenant le sous-formulaire.

Cette commande est utile par exemple dans le cas où des objets du sous-formulaire doivent être dimensionnés en fonction des caractéristiques de l'objet sous-formulaire lui-même. Dans l'événement formulaire Sur chargement, le sous-formulaire peut appeler cette commande pour calculer la place dont il dispose afin d'afficher son contenu.

Note : Il n'est pas possible d'utiliser directement l'événement Sur redimensionnement dans la méthode du sous-formulaire car cet événement est lié au redimensionnement d'une fenêtre. Il est donc généré uniquement dans la méthode du formulaire parent. Vous pouvez cependant, depuis cet événement du formulaire parent, appeler explicitement le sous-formulaire par exemple via la commande **EXECUTER METHODE DANS SOUS FORMULAIRE**.

- Si la commande est appelée depuis un formulaire qui n'est pas en cours d'utilisation en tant que sous-formulaire, elle retourne la taille courante de la fenêtre du formulaire.
- Si la commande est appelée en-dehors du contexte de l'affichage l'écran (par exemple lors de l'impression du formulaire), elle retourne 0 dans *largeur* et *hauteur*.

OBJET Lire texte exemple

OBJET Lire texte exemple ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
Résultat	Texte	⇐	Texte d'exemple associé à l'objet

Description

La commande **OBJET Lire texte exemple** retourne le texte d'exemple associé à l'objet ou aux objets désigné(s) par les paramètres *objet* et *. Si aucun texte d'exemple n'est associé à l'objet, la commande retourne une chaîne vide.

Un texte d'exemple peut avoir été défini soit via la Liste des propriétés, soit via la commande **OBJET FIXER TEXTE EXEMPLE**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Si le texte d'exemple est une référence xliff définie via la Liste des propriétés, la commande retourne la référence d'origine sous la forme ":xliff:*resname*" et non sa valeur calculée.

Exemple

Vous souhaitez lire le texte exemple d'un champ :

```
$txt:=OBJET Lire texte exemple([Personnes]Nom)
```

OBJET Lire titre

OBJET Lire titre ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
Résultat	Texte	⇐	Libellé de l'objet

Description

La commande **OBJET Lire titre** retourne le titre (libellé) du ou des objet(s) de formulaire désigné(s) par *objet*.

OBJET Lire titre peut être utilisée avec tous les types d'objets simples contenant un libellé :

- boutons,
- cases à cocher,
- boutons radio,
- textes statiques,
- zones de groupe.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

OBJET Lire type

OBJET Lire type ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	⇐	Type d'objet

Description

La commande **OBJET Lire type** retourne le type de l'objet désigné par le(s) paramètre(s) *objet* et * dans le formulaire courant.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Cette syntaxe est obligatoire si vous traitez des objets statiques tels que des lignes ou des rectangles.
Si vous ne passez pas ce paramètre, vous indiquez que le paramètre *objet* est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable.

Note : Si vous appliquez la commande à un ensemble d'objets, le type du dernier objet est retourné.

La valeur retournée correspond à l'une des constantes suivantes du thème "**Types objets formulaire**" :

Constante	Type	Valeur
Objet type bouton 3D	Entier long	16
Objet type bouton image	Entier long	19
Objet type bouton inversé	Entier long	17
Objet type bouton invisible	Entier long	18
Objet type bouton poussoir	Entier long	15
Objet type bouton radio	Entier long	22
Objet type bouton radio 3D	Entier long	23
Objet type bouton radio image	Entier long	24
Objet type cadran	Entier long	28
Objet type case à cocher	Entier long	25
Objet type case à cocher 3D	Entier long	26
Objet type champ radio boutons	Entier long	5
Objet type combobox	Entier long	11
Objet type grille de boutons	Entier long	20
Objet type groupe	Entier long	21
Objet type image statique	Entier long	2
Objet type inconnu	Entier long	0
Objet type indicateur de progression	Entier long	27
Objet type ligne	Entier long	32
Objet type listbox	Entier long	7
Objet type listbox colonne	Entier long	9
Objet type listbox entête	Entier long	8
Objet type listbox pied	Entier long	10
Objet type liste hiérarchique	Entier long	6
Objet type matrice	Entier long	35
Objet type menu déroulant hiérarchique	Entier long	13
Objet type onglet	Entier long	37
Objet type ovale	Entier long	34
Objet type popup liste déroulante	Entier long	12
Objet type popup menu image	Entier long	14
Objet type rectangle	Entier long	31
Objet type rectangle arrondi	Entier long	33
Objet type règle	Entier long	29
Objet type saisie image	Entier long	4
Objet type saisie texte	Entier long	3
Objet type séparateur	Entier long	36
Objet type sous-formulaire	Entier long	39
Objet type texte statique	Entier long	1
Objet type zone de groupe	Entier long	30
Objet type zone plug-in	Entier long	38
Objet type zone web	Entier long	40
Objet type zone write pro	Entier long	41

Exemple

Vous souhaitez charger un formulaire et obtenir la liste de tous les objets des list box qu'il contient.

```
FORM CHARGER("MonFormulaire")
TABLEAU TEXTE(tabObjets;0)
```

```
FORM LIRE OBJETS(tabObjets)
TABLEAU ENTIER LONG(ar_type;Taille tableau(tabObjets))
Boucle($i;1;Taille tableau(tabObjets))
      ar_type{$i}:=OBJET Lire type(*;tabObjets{$i})
      Si(ar_type{$i}=Objet type listbox)
         TABLEAU TEXTE(tabObjetsLB;0)
         LISTBOX LIRE OBJETS(*;tabObjets{$i};tabObjetsLB)
      Fin de si
Fin de boucle
FORM LIBERER
```

OBJET Lire type indicateur

OBJET Lire type indicateur ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	⇐	Type d'indicateur

Description

La commande **OBJET Lire type indicateur** retourne le type d'indicateur courant affecté au(x) thermomètre(s) désigné(s) par le(s) paramètre(s) *objet* et *.

Le type d'indicateur peut être défini via la Liste des propriétés en mode Développement ou via la commande **OBJET FIXER TYPE INDICATEUR**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Vous pouvez comparer la valeur retournée par la commande aux constantes suivantes, placées dans le thème "**Objets de formulaire (Propriétés)**" :

Constante	Type	Valeur	Comment
Indicateur Barber shop	Entier long	2	Barre affichant une animation continue
Indicateur Barre de progression	Entier long	1	Barre de progression standard
Indicateur de progression asynchrone	Entier long	3	Indicateur circulaire affichant une animation continue

OBJET LIRE VALEUR MAXIMUM

OBJET LIRE VALEUR MAXIMUM ({* ;} objet ; valeurMaxi)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
valeurMaxi	Date, Heure, Numérique	⇐	Valeur maximale actuelle de l'objet

Description

La commande **OBJET LIRE VALEUR MAXIMUM** retourne dans la variable *valeurMaxi* la valeur maximum actuelle de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

La propriété "Valeur maximum" peut être définie via la Liste des propriétés en mode Développement ou via la commande **OBJET FIXER VALEUR MAXIMUM**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

OBJET LIRE VALEUR MINIMUM

OBJET LIRE VALEUR MINIMUM ({* ;} objet ; valeurMini)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
valeurMini	Date, Heure, Numérique	⇐	Valeur minimale actuelle de l'objet

Description

La commande **OBJET LIRE VALEUR MINIMUM** retourne dans la variable *valeurMini* la valeur minimum courante de l'objet ou des objets désigné(s) par les paramètres *objet* et *.

La propriété "Valeur minimum" peut être définie via la Liste des propriétés en mode Développement ou via la commande **OBJET FIXER VALEUR MINIMUM**.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

OBJET Lire visible

OBJET Lire visible ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
Résultat	Booléen	⇐	Vrai = objet(s) visible(s), Faux sinon

Description

La commande **OBJET Lire visible** retourne Vrai si l'objet ou le groupe d'objets désigné(s) par *objet* dispose de l'attribut visible et Faux sinon.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

_o_ACTIVER BOUTON

_o_ACTIVER BOUTON ({* ;} objet)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Note de compatibilité

La commande **_o_ACTIVER BOUTON** est déclarée obsolète dans 4D à compter de la version 12 et est conservée pour des raisons de compatibilité uniquement. Sa portée globale, incluant toutes les instances de la variable désignée et non uniquement celles du formulaire courant, ne correspond pas à celle des commandes du thème "Objets (Formulaires)".
_o_ACTIVER BOUTON est avantageusement remplacée par la commande **OBJET FIXER ACTIVATION**.

_o_INACTIVER BOUTON

_o_INACTIVER BOUTON ({* ;} objet)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Note de compatibilité

La commande **_o_INACTIVER BOUTON** est déclarée obsolète dans 4D depuis la version 12 et est conservée pour des raisons de compatibilité uniquement. Sa portée globale, incluant toutes les instances de la variable désignée et non uniquement celles du formulaire courant, ne correspond pas à celle des commandes du thème "Objets (Formulaires)".
_o_INACTIVER BOUTON est avantageusement remplacée par la commande **OBJET FIXER ACTIVATION**.

Objets (Langage)

- Structure des objets de langage 4D
- OB Copier
- OB Est defini
- OB Est vide
- OB FIXER
- OB FIXER NULL
- OB FIXER TABLEAU
- OB Lire
- OB LIRE NOMS PROPRIETES
- OB LIRE TABLEAU
- OB Lire type
- OB SUPPRIMER

Structure des objets de langage 4D

Les commandes du thème **Objets (Langage)** permettent de créer et de manipuler des données sous forme d'objet. Cette fonctionnalité étend les possibilités d'échange entre 4D et tout type d'application prenant en charge les données structurées.

Toutes les commandes de ce thème prennent en charge les objets 4D suivants :

- les variables objet ou tableaux objet créé(e)s et initialisé(e)s à l'aide des commandes **C_OBJET** (thème "**Compilateur**") ou **TABLEAU OBJET** (thème "**Tableaux**").
- les champs objet provenant de la base de données 4D (cf. **Types de champs 4D**).

La structure des objets "natifs" 4D est basée sur le principe classique des paires "attribut/valeur". La syntaxe de ces objets s'inspire de la notation JSON, mais ne la suit pas entièrement.

Note : Pour manipuler des objets JSON, vous devez utiliser les commandes du thème "**JSON**".

- Un **nom** d'attribut est toujours un texte, par exemple "Nom".
- Une **valeur** d'attribut peut être du type suivant :
  - nombre (réel, entier long, etc.)
  - texte
  - tableau (texte, réel, entier long, entier, booléen, objet, pointeur)
  - null
  - booléen
  - date (format "YYYY-MM-DDTHH:mm:ss.SSSZ")
  - objet (les objets peuvent être imbriqués sur plusieurs niveaux)
  - les variables objet et tableaux objet acceptent également des valeurs de type pointeur (stocké tel quel, évalué à l'aide de la commande **JSON Stringify** ou lors d'une copie).

Attention : Gardez à l'esprit que les noms d'attributs différencient les majuscules/minuscules.

OB Copier

OB Copier (objet {; résoudrePtrs}) -> Résultat

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
résoudrePtrs	Booléen	➡	Vrai = résoudre les pointeurs, Faux ou omis = ne pas les résoudre
Résultat	Objet	⬅	Copie de objet

Description

La commande **OB Copier** retourne un objet contenant une copie complète des propriétés, sous-objets et valeurs de *objet*.

objet doit avoir été défini via la commande **C_OBJET** ou désigner un champ objet 4D.

Si *objet* contient des valeurs de type pointeur, par défaut la copie contient également les pointeurs. Vous pouvez cependant résoudre les pointeurs au moment de la copie : pour cela, passez **Vrai** dans le paramètre *résoudrePtrs*. Dans ce cas, chaque pointeur présent comme valeur dans objet sera évalué au moment de la copie et sa valeur dépointée sera utilisée.

Exemple 1

Vous souhaitez dupliquer un objet contenant des valeurs simples :

```
C_OBJET($Object)
C_TEXTE($JsonString)

TABLEAU OBJET($arraySel;0)
TOUT SELECTIONNER([Product])
Tant que(Non(Fin de selection([Product])))
   OB FIXER($Object;"id";[Product]ID_Product)
   OB FIXER($Object;"Product Name";[Product]Product_Name)
   OB FIXER($Object;"Price";[Product]Price)
   OB FIXER($Object;"Tax rate";[Product]Tax_rate)
   $ref_value:=OB Copier($Object) //copie directe
   AJOUTER A TABLEAU($arraySel;$ref_value)
         //le contenu de $ref_value est identique à celui de $Object
   ENREGISTREMENT SUIVANT([Product])
Fin tant que
    //le contenu de $ref_value
$JsonString:=JSON Stringify tableau($arraySel)
```

Exemple 2

Vous dupliquez un objet contenant des pointeurs :

```
C_OBJET($ref)

OB FIXER($ref;"name";->[Company]name) //objet avec pointeurs
OB FIXER($ref;"country";->[Company]country)
TABLEAU OBJET($sel;0)
TABLEAU OBJET($sel2;0)

TOUT SELECTIONNER([Company])

Tant que(Non(Fin de selection([Company])))
      $ref_comp:=OB Copier($ref) // copie sans évaluation des pointeurs
          //$ref_comp={"name":"->[Company]name","country":"->[Company]Country"}
      $ref_comp2:=OB Copier($ref;Vrai) //copie avec évaluation des pointeurs
          //$ref_comp={"name":"4D SAS","country":"France"}
      AJOUTER A TABLEAU($sel;$ref_comp)
      AJOUTER A TABLEAU($sel2;$ref_comp2)
      ENREGISTREMENT SUIVANT([Company])
Fin tant que

$Object:=JSON Stringify tableau($sel)
$Object2:=JSON Stringify tableau($sel2)

    // $Object = [{"name":"","country":""},{"name":"","country":""},...]
    // $Object2 = [{"name":"4D SAS","country":"France"},{"name":"4D, Inc","country":"USA"},
{"name":"Catalan","country":"France"}...]
```

OB Est defini

OB Est defini (objet {; propriété}) -> Résultat

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
propriété	Texte	➡	Si passé = propriété à vérifier, si omis = vérifier l'objet
Résultat	Booléen	⮌	Si propriété omis : Vrai si objet est défini, sinon Faux. Si propriété passé : Vrai si propriété est définie, sinon Faux

Description

La commande **OB Est defini** retourne **Vrai** si *objet* ou *propriété* est défini, et **Faux** sinon.

objet doit avoir été créé via la commande **C_OBJET** ou désigner un champ objet 4D.

Par défaut, si vous omettez le paramètre *propriété*, la commande vérifie que *objet* est défini. Un objet est défini si son contenu a été initialisé.

Note : Un objet peut être défini mais vide. Pour savoir si un objet est indéfini ou vide, utilisez la commande **OB Est vide**.

Si vous passez le paramètre *propriété*, la commande vérifie si cette propriété existe dans *objet*. Attention, le paramètre *propriété* tient compte des majuscules/minuscules.

Exemple 1

Syntaxe testant l'initialisation d'un objet :

```
C_OBJET($objet)
$def:=OB Est defini($objet) //$def=faux car $objet n'est pas initialisé

OB FIXER($objet;"nom";"Martin")
OB SUPPRIMER($objet;"nom")
$def2:=OB Est defini($objet) //$def2=vrai car $objet est vide {} mais a été initialisé
```

Exemple 2

Test de l'existence d'une propriété :

```
C_OBJET($ref)
OB FIXER($ref;"nom";"smith";"age";42)
    //...
Si(OB Est defini($ref;"age"))
          //...
Fin de si
```

Ce test équivaut à :

```
Si(OB Lire type($Objet;"nom")#Est une variable indéfinie)
```

OB Est vide

OB Est vide (objet) -> Résultat

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
Résultat	Booléen	⮌	Vrai si objet est vide ou indéfini, sinon Faux

Description

La commande **OB Est vide** retourne **Vrai** si *objet* est indéfini ou vide, et **Faux** si *objet* est défini (initialisé) et contient au moins une propriété.
objet doit avoir été créé via la commande **C_OBJET** ou désigner un champ objet 4D.

Exemple

Voici les différents résultats de la commande ainsi que de la commande **OB Est defini**, en fonction du contexte :

```
C_OBJET($ref)
$vide:=OB Est vide($ref) //Vrai
$def:=OB Est defini($ref) //Faux

OB FIXER($ref;"nom";"Susie";"age";4)
    //$ref="{"nom":"Susie","age":4}"
$vide:=OB Est vide($ref) //Faux
$def:=OB Est defini($ref) //Vrai

OB SUPPRIMER($ref;"nom")
OB SUPPRIMER($ref;"age")
$vide:=OB Est vide($ref) //Vrai
$def:=OB Est defini($ref) //Vrai
```

OB FIXER

OB FIXER (objet ; propriété ; valeur {; propriété2 ; valeur2 ; ... ; propriétéN ; valeurN})

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
propriété	Texte	➡	Nom de la propriété à définir
valeur	Texte, Date, Booléen, Pointeur, Numérique, Objet	➡	Nouvelle valeur de la propriété

Description

La commande **OB FIXER** permet de créer ou de modifier une ou plusieurs paires propriété/valeur dans l'objet de langage désigné par le paramètre *objet*.

objet doit avoir été défini via la commande **C_OBJET** ou désigner un champ objet 4D.

Note : Cette commande prend en charge les définitions d'attributs dans les *objets* 4D Write Pro, comme la commande **WP FIXER ATTRIBUTS** (cf. exemple 10). Toutefois, à la différence de **WP FIXER ATTRIBUTS**, **OB FIXER** ne permet pas de manipuler directement une variable ou un champ image comme valeur d'attribut.

Passez dans le paramètre *propriété* le libellé de la propriété à créer ou à modifier. Si la propriété existe déjà dans *objet*, sa valeur sera mise à jour. Si elle n'existe pas, elle est créée.
Attention, le paramètre *propriété* tient compte des majuscules/minuscules.

Passez dans le paramètre *valeur* la valeur de la propriété à définir. Plusieurs types de données sont pris en charge. A noter que :

- si vous passez un pointeur, il est conservé tel quel, il est évalué à l'aide de la commande **JSON Stringify**,
- les dates sont stockées au format "YYYY-MM-DDTHH:mm:ss.SSSZ". Lors de la conversion d'une date 4D en texte avant stockage dans l'objet, par défaut le programme tient compte du fuseau horaire local. Vous pouvez modifier ce fonctionnement à l'aide du sélecteur JSON fuseau horaire local de la commande **FIXER PARAMETRE BASE**.
- si vous passez une heure, elle est stockée sous la forme d'un nombre de millisecondes (réel) dans *objet*.
- si vous passez un objet de langage, la commande utilise la référence de l'objet et non une copie.

Exemple 1

Création d'un objet et ajout d'une propriété de type texte :

```
C_OBJET($Object)
OB FIXER($Object ;"prénom";"John";"nom";"Smith")
    //$Object = {"prénom":"John","nom":"Smith"}
```

Exemple 2

Création d'un objet et ajout d'une propriété de type booléen :

```
C_OBJET($Object)
OB FIXER($Object ;"nom";"smith";"age";42;"client";Vrai)
    //$Object = {"nom":"smith","age":42,"client":true}
```

Exemple 3

Modification d'une propriété :

```
    //$Object = {"prénom":"John","nom":"Smith"}
OB FIXER($Object ;"prénom";"Paul")
    //$Object = {"prénom":"Paul","nom":"Smith"}
```

Exemple 4

Ajout d'une propriété :

```
    //$Object = {"prénom":"John","nom":"Smith"}
OB FIXER($Object ;"service";"Comptabilité")
    //$Object = {"prénom":"Paul","nom":"Smith","service":"Comptabilité"}
```

Exemple 5

Renommage d'une propriété :

```
C_OBJET($Object)
OB FIXER($Object ;"nom";"James";"age";35)
    //$Object = {"nom":"James","age":35}
OB FIXER($Object ;"prénom";OB Lire($Object ;"nom"))
```

```
    //$Object = {"prénom":""James","nom":"James","age":35}
OB SUPPRIMER($Object ;"nom")
    //$Object = {"prénom":""James","age":35}
```

Exemple 6

Utilisation d'un pointeur :

```
    //$Object = {"prénom":"Paul","nom":"Smith"}
C_TEXTE($nom)
OB FIXER($Object ;"nom";->$nom)
    //$Object = {"prénom":"Paul","nom":"->$nom"}
$JsonString:=JSON Stringify($Object)
    //$JsonString="{"prénom":"Paul","nom":""}
$nom:="Wesson"
$JsonString:=JSON Stringify($Object)
    //$JsonString="{"prénom":"Paul","nom":"Wesson"}
```

Exemple 7

Utilisation d'un objet :

```
C_OBJET($ref_smith)
OB FIXER($ref_smith ;"nom";"Smith")
C_OBJET($ref_emp)
OB FIXER($ref_emp ;"employé";$ref_smith)
$Json_string :=JSON Stringify($ref_emp)
    // $ref_emp = {"employé":{"nom":"Smith"}} (objet)
    // $Json_string = "{"employé":{"nom":"Smith"}}" (chaîne)
```

Vous pouvez également changer une valeur à la volée :

```
OB FIXER($ref_smith ;"nom";"Smyth")
    // $ref_smith = {"employé":{"nom":"Smyth"}}
$string :=JSON Stringify($ref_emp)
    // $string = "{"employee":{"nom":"Smyth"}}"
```

Exemple 8

Si vous avez défini le champ [Rect]Desc en tant que champ objet, vous pouvez écrire :

```
CREER ENREGISTREMENT([Rect])
[Rect]Name:="Blue square"
OB FIXER([Rect]Desc;"x";"50";"y";"50";"color";"blue")
STOCKER ENREGISTREMENT([Rect])
```

Exemple 9

Vous souhaitez exporter des données en JSON contenant une date 4D convertie. A noter que la conversion a lieu au moment du stockage de la date dans l'objet, il faut donc appeler la commande **FIXER PARAMETRE BASE** avant **OB FIXER** :

```
C_OBJET($o)
FIXER PARAMETRE BASE(JSON fuseau horaire local;0)
OB FIXER($o ;"maDate";Date du jour) // conversion JSON
$json:=JSON Stringify($o)
FIXER PARAMETRE BASE(JSON fuseau horaire local;1)
```

Exemple 10

Dans la méthode d'un formulaire contenant une zone 4D Write Pro, vous pouvez écrire :

```
Si(Evenement formulaire=Sur validation)
   OB FIXER([MyDocuments]My4DWP;"myatt_Last edition by";Utilisateur courant)
   OB FIXER([MyDocuments]My4DWP;"myatt_Category";"Memo")
Fin de si
```

Vous pouvez également lire les attributs personnalisés des documents :

```
vAttrib:=OB Lire([MyDocuments]My4DWP;"myatt_Last edition by")
```

OB FIXER NULL

OB FIXER NULL (objet ; propriété)

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
propriété	Texte	➡	Nom de la propriété à laquelle appliquer la valeur null

Description

La commande **OB FIXER NULL** permet de stocker la valeur **null** dans l'objet de langage désigné par le paramètre *objet*.

objet doit avoir été défini via la commande **C_OBJET** ou désigner un champ objet 4D.

Passez dans le paramètre *propriété* le libellé de la propriété dans laquelle stocker la valeur **null**. Si la propriété existe déjà dans *objet*, sa valeur sera mise à jour. Si elle n'existe pas, elle est créée. Attention, le paramètre *propriété* tient compte des majuscules/minuscules.

Exemple

On souhaite mettre la valeur null dans la propriété "âge" de Léa :

```
C_OBJET($ref)
OB FIXER($ref;"nom";"Léa";"âge";4)
    // $ref = {"nom":"Léa","âge":4}
...
OB FIXER NULL($ref ;"âge")
    // $ref = {"nom":"Léa","âge":null}
```

OB FIXER TABLEAU

OB FIXER TABLEAU (objet ; propriété ; tableau)

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
propriété	Texte	➡	Nom de la propriété à définir
tableau	Tableau texte, Tableau réel, Tableau booléen, Tableau objet, Tableau pointeur, Tableau entier long	➡	Tableau à stocker dans la propriété

Description

La commande **OB FIXER TABLEAU** permet de définir le *tableau* à associer à la *propriété* dans l'objet de langage désigné par le paramètre *objet*.

objet doit avoir été défini via la commande **C_OBJET** ou désigner un champ objet 4D.

Passez dans le paramètre *propriété* le libellé de la propriété à créer ou à modifier. Si la propriété existe déjà dans *objet*, sa valeur sera mise à jour. Si elle n'existe pas, elle est créée. Attention, le paramètre *propriété* tient compte des majuscules/minuscules.

Passez dans le paramètre *tableau* le tableau devant être passé comme valeur de la propriété. Plusieurs types de tableaux sont pris en charge.

Note : Il n'est pas possible d'utiliser de tableaux à deux dimensions.

Exemple 1

Utilisation d'un tableau texte :

```
C_OBJET($Children)
TABLEAU TEXTE($tabChildren;3)
$tabChildren{1}:="Richard"
$tabChildren{2}:="Susan"
$tabChildren{3}:="James"

OB FIXER TABLEAU($Children;"Children";$tabChildren)
    // Valeur de $Children = {"Children":["Richard","Susan","James"]}
```

Exemple 2

Ajout d'un élément dans un tableau :

```
TABLEAU TEXTE($tabText;2)
$tabText{1}:="Smith"
$tabText{2}:="White"
C_OBJET($Employees)
OB FIXER TABLEAU($Employees;"Employés";$tabText)
AJOUTER A TABLEAU($tabText;"Brown") //Ajout dans le tableau 4D
    // $Employees = {"Employés":["Smith","White"]}

OB FIXER TABLEAU($Employees;"Employés";$tabText)
    // $Employees = {"Employés":["Smith","White","Brown"]}
```

Exemple 3

Utilisation d'un tableau texte avec sélection d'un élément :

```
    //$Employees = {"Employés":["Smith","White","Brown"]}
OB FIXER TABLEAU($Employees ;"Manager";$tabText{1})
    //$Employees = {"Employees":["Smith","White","Brown"],"Manager":["Smith"]}
```

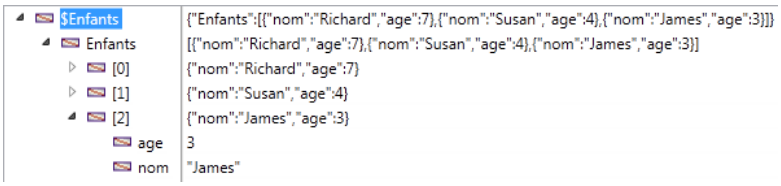
Exemple 4

Utilisation d'un tableau objet :

```
C_OBJET($Enfants;$ref_richard;$ref_susan;$ref_james)
TABLEAU OBJET($tabEnfants;0)
OB FIXER($ref_richard;"nom";"Richard";"age";7)
AJOUTER A TABLEAU($tabEnfants;$ref_richard)
OB FIXER($ref_susan;"nom";"Susan";"age";4)
AJOUTER A TABLEAU($tabEnfants;$ref_susan)
OB FIXER($ref_james;"nom";"James";"age";3)

AJOUTER A TABLEAU($tabEnfants;$ref_james)

    //$tabEnfants {1} = {"nom":"Richard","age":7}
```

```
      //$tabEnfants {2} = {"nom":"Susan","age":4}
      //$tabEnfants {3} = {"nom":"James","age":3}

   OB FIXER TABLEAU($Enfants;"Enfants";$tabEnfants)

      // $Enfants = {"Enfants":[{"nom":"Richard","age":7},{"nom":"Susan",
      // "age":4},{"nom":"James","age":3}]}
```

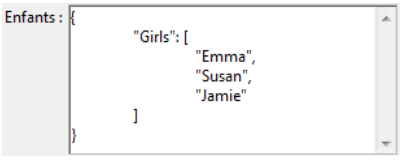
L'objet est représenté ainsi dans le débogueur :



Exemple 5

Utilisation d'un champ objet :

```
   TABLEAU TEXTE($arrGirls;3)
   $arrGirls{1}:="Emma"
   $arrGirls{2}:="Susan"
   $arrGirls{3}:="Jamie"
   OB FIXER TABLEAU([Personnes]Enfants;"Girls";$arrGirls)
```



OB Lire

OB Lire (objet ; propriété {; type}) -> Résultat

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
propriété	Texte	➡	Nom de la propriété à lire
type	Entier long	➡	Type dans lequel convertir la valeur
Résultat	Booléen, Date, Objet, Pointeur, Réel, Texte	⬅	Valeur courante de la propriété

Description

La commande **OB Lire** retourne la valeur courante de la *propriété* de l'*objet*, convertie optionnellement dans le *type* défini.

objet doit avoir été défini via la commande **C_OBJET** ou désigner un champ objet 4D.

Note : Cette commande prend en charge les définitions d'attributs dans les *objets* 4D Write Pro, comme la commande **WP LIRE ATTRIBUTS** (cf. exemple 9). Toutefois, à la différence de **WP LIRE ATTRIBUTS**, **OB Lire** ne permet pas de manipuler directement une variable ou un champ image comme valeur d'attribut.

Passez dans le paramètre *propriété* le libellé de la propriété à lire. Attention, le paramètre *propriété* tient compte des majuscules/minuscules.

Par défaut, 4D retournera la valeur de la propriété dans son type d'origine. Vous pouvez "forcer" le typage de la valeur retournée à l'aide du paramètre optionnel *type*. Pour cela, vous pouvez passer dans *type* une des constantes suivantes, placées dans le thème **Types champs et variables** :

Constante	Type	Valeur
Est un booléen	Entier long	6
Est un entier long	Entier long	9
Est un numérique	Entier long	1
Est un objet	Entier long	38
Est un pointeur	Entier long	23
Est un texte	Entier long	2
Est une date	Entier long	4
Est une heure	Entier long	11

La commande retourne la valeur de la *propriété*. Plusieurs types de données sont pris en charge. A noter que :

- un pointeur est retourné tel quel, il peut être évalué à l'aide de la commande **JSON Stringify**,
- les dates sont retournées au format "YYYY-MM-DDTHH:mm:ss.SSSZ"
- dans les valeurs réelles, le séparateur décimal est toujours le point "."
- les heures sont retournées sous forme d'un nombre. A noter que **OB FIXER** stocke les heures sous forme de millisecondes, conformément au standard javascript, tandis que 4D attend un nombre de secondes. Pour une interprétation correcte par **OB Lire** d'une heure stockée, vous devez utiliser la constante Est une heure.

Exemple 1

Récupération d'une valeur de type texte :

```
C_OBJET($ref)
C_TEXTE($prénom)
OB FIXER($ref;"Prénom";"Harry")
$prénom:=OB Lire($ref;"Prénom")  // $prénom = "Harry" (texte)
```

Exemple 2

Récupération d'une valeur numérique convertie en entier long :

```
OB FIXER($ref ;"age";42)
$age:=OB Lire($ref ;"age") // $age est un réel (défaut)
$age:=OB Lire($ref ;"age";Est un entier long) // $age est un entier long
```

Exemple 3

Récupération des valeurs d'un objet :

```
C_OBJET($ref1;$ref2)
OB FIXER($ref1;"nom";"Smith") //$ref1={"nom":"Smith"}
OB FIXER($ref2;"fils";$ref1) //$ref2={"fils":{"nom":"Smith"}}
$fils:=OB Lire($ref2;"fils") //$fils={"name":"john"} (objet)
$nomfils:=OB Lire($fils ;"nom") //$nomfils="john" (texte)
```

Exemple 4

Modifications de l'âge d'un employé :

```
C_OBJET($ref_john;$ref_jim)
OB FIXER($ref_john;"nom";"John";"age";35)
OB FIXER($ref_jim;"nom";"Jim";"age";40)
AJOUTER A TABLEAU($myArray;$ref_john) // on crée un tableau objet
AJOUTER A TABLEAU($myArray;$ref_jim)
    // on passe l'âge de John de 35 à 25
OB FIXER($myArray{1};"age";25)
    // On remplace l'âge de "John" dans le tableau
Boucle($i;1;Taille tableau($myArray))
      Si(OB Lire($myArray{$i};"nom")="John")
         OB FIXER($myArray{$i};"age";36)  //au lieu de 25
            // $ref_john={"nom":"John","age":36}
      Fin de si
Fin de boucle
```

Exemple 5

Désérialisation d'une chaîne date formatée en ISO :

```
C_OBJET($object)
C_DATE($anniv)
C_TEXTE($chainAnniv)
OB FIXER($object;"Anniversaire";"1990-12-25T12:00:00Z")
$chainAnniv:=OB Lire($object;"Anniversaire")
    // $chainAnniv="1990-12-25T12:00:00Z"
$anniv:=OB Lire($object;"Anniversaire";Est une date)
    //$anniv=25/12/90
```

Exemple 6

Utilisation d'objets imbriqués :

```
C_OBJET($ref1;$child;$children)
C_TEXTE($childName)
OB FIXER($ref1;"firstname";"John";"lastname";"Monroe")
    //{"firstname":"john","lastname";"Monroe"}
OB FIXER($children;"children";$ref1)
$child:=OB Lire($children;"children")
    //$son = {"firstname":"John","lastname":"Monroe"} (objet)
$childName:=OB Lire($child;"lastname")
    //$childName = "Monroe" (texte)
    //ou bien
$childName:=OB Lire(OB Lire($children;"children");"lastname")
    // $childName = "Monroe" (texte)
```

Exemple 7

Récupération dans 4D d'une heure stockée dans un objet :

```
C_OBJET($obj_o)
C_HEURE($set_h;$get_h)

$set_h:=?01:00:00?+1
OB FIXER($obj_o;"myHour";$set_h)
 // $obj_o == {"myHour":3601000}
 // L'heure est stockée en millisecondes.
$get_h:=OB Lire($obj_o;"myHour";Est une heure)
 // $get_h == ?01:00:01?
 // L'heure est correctement lue
```

Exemple 8

Exemples de manipulation de champs objet 4D :

```
 // Définir une valeur
OB FIXER([Personnes]Identity_OB;"Prénom";$firstName)
OB FIXER([Personnes]Identity_OB;"Nom";$lastName)

 // Lire une valeur
$firstName:=OB Lire([Personnes]Identity_OB;"Prénom")
$lastName:=OB Lire([Personnes]Identity_OB;"Nom")
```

Exemple 9

Dans la méthode d'un formulaire contenant une zone 4D Write Pro, vous pouvez écrire :

```
Si(Evenement formulaire=Sur validation)
   OB FIXER([MyDocuments]My4DWP;"myatt_Last edition by";Utilisateur courant)
   OB FIXER([MyDocuments]My4DWP;"myatt_Category";"Memo")
Fin de si
```

Vous pouvez également lire les attributs personnalisés des documents :

```
vAttrib:=OB Lire([MyDocuments]My4DWP;"myatt_Last edition by")
```

OB LIRE NOMS PROPRIETES

OB LIRE NOMS PROPRIETES (objet ; tabPropriétés {; tabTypes})

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
tabPropriétés	Tableau texte	⬅	Noms des propriétés
tabTypes	Tableau entier long	⬅	Types des propriétés

Description

La commande **OB LIRE NOMS PROPRIETES** retourne dans *tabPropriétés* les noms des propriétés contenues dans l'objet de langage désigné par le paramètre *objet*.

objet doit avoir été défini via la commande **C_OBJET** ou désigner un champ objet 4D.

Passez dans le paramètre *tabPropriétés* un tableau texte. Si le tableau n'existe pas, la commande le crée et le dimensionne automatiquement.

Optionnellement, vous pouvez passer dans *tabTypes* un tableau entier long. Pour chaque élément de *tabPropriétés*, la commande retournera dans *tabTypes* le type de la valeur stockée dans la propriété. Vous pouvez comparer les valeurs reçues aux constantes suivantes du thème **Types champs et variables** :

Constante	Type	Valeur
Est un booléen	Entier long	6
Est un null JSON	Entier long	255
Est un numérique	Entier long	1
Est un objet	Entier long	38
Est un tableau objet	Entier long	39
Est un texte	Entier long	2

Exemple 1

Vous souhaitez tester qu'un objet n'est pas vide :

```
TABLEAU TEXTE(tabNoms;0)
TABLEAU ENTIER LONG(tabTypes;0)
C_OBJET($ref_richard)
OB FIXER($ref_richard;"nom";"Richard";"age";7)
OB LIRE NOMS PROPRIETES($ref_richard;tabNoms;tabTypes)
    //tabNoms{1}="nom", tabNoms{2}="age"
    //tabTypes{1}=2, tabTypes{2}=1
Si(Taille tableau(tabNoms)#0)
      //...
Fin de si
```

Exemple 2

Utilisation d'un élément de tableau d'objets :

```
C_OBJET($Children;$ref_richard;$ref_susan;$ref_james)
TABLEAU OBJET($arrayChildren;0)

OB FIXER($ref_richard;"nom";"Richard";"age";7)
AJOUTER A TABLEAU($arrayChildren;$ref_richard)
OB FIXER($ref_susan;"nom";"Susan";"age";4;"fille";Vrai) //attribut supplémentaire
AJOUTER A TABLEAU($arrayChildren;$ref_susan)
OB FIXER($ref_james;"nom";"James")
OB FIXER NULL($ref_james;"age") //attribut null
AJOUTER A TABLEAU($arrayChildren;$ref_james)

OB LIRE NOMS PROPRIETES($arrayChildren{1};$tabNoms;$tabTypes)
 // $arrayChildren{1} = {"nom":"Richard","age":7}
 // $tabNoms{1}="nom"
 // $tabNoms{2}="age"
 // $tabTypes{1}=2
 // $tabTypes{2}=1

OB LIRE NOMS PROPRIETES($arrayChildren{2};$tabNoms;$tabTypes)
 // $arrayChildren{3} = {"nom":"Susan","age":4,"fille":true}
 // $tabNoms{1}="nom"
 // $tabNoms{2}="age"
 // $tabNoms{3}="fille"
 // $tabTypes{1}=2
 // $tabTypes{2}=1
 // $tabTypes{3}=6
```

```
OB LIRE NOMS PROPRIETES($arrayChildren{3};$tabNoms;$tabTypes)
 // $arrayChildren{3} = {"nom":"James","age":null}
 // $tabNoms{1}="nom"
 // $tabNoms{2}="age"
 // $tabTypes{1}=2
 // $tabTypes{2}=255
```

OB LIRE TABLEAU

OB LIRE TABLEAU (objet ; propriété ; tableau)

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
propriété	Texte	➡	Nom de la propriété à lire
tableau	Tableau texte, Tableau réel, Tableau booléen, Tableau objet, Tableau pointeur, Variable entier long	⬅	Tableau valeur de la propriété

Description

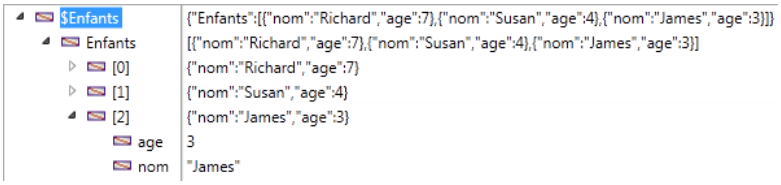
La commande **OB LIRE TABLEAU** récupère dans *tableau* le tableau de valeurs stocké dans la *propriété* de l'objet de langage désigné par le paramètre *objet*.

objet doit avoir été défini via la commande **C_OBJET** ou désigner un champ objet 4D.

Passez dans le paramètre *propriété* le libellé de la propriété à lire. Attention, le paramètre *propriété* tient compte des majuscules/minuscules.

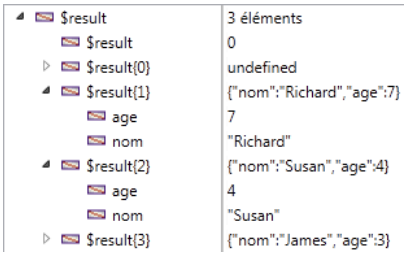
Exemple 1

Soit le tableau objet défini dans l'exemple de la commande **OB FIXER TABLEAU** :



On souhaite récupérer ces valeurs :

```
TABLEAU OBJET($result;0)
OB LIRE TABLEAU($Enfants;"Enfants";$result)
```



Exemple 2

On souhaite changer une valeur dans le premier élément du tableau :

```
    //Changer la valeur de "age" :
TABLEAU OBJET($refs)
OB LIRE TABLEAU($refEmployees;"__ENTITIES";$refs)
OB FIXER($refs{1};"age";25)
```

OB Lire type

OB Lire type (objet ; propriété) -> Résultat

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
propriété	Texte	➡	Nom de la propriété
Résultat	Entier long	⬅	Type de valeur de la propriété

Description

La commande **OB Lire type** retourne le type de la valeur associée à la *propriété* de l'*objet*.

objet doit avoir été défini via la commande **C_OBJET** ou désigner un champ objet 4D.

Passez dans le paramètre *propriété* le libellé de la propriété dont vous souhaitez connaître le type. Attention, le paramètre *propriété* tient compte des majuscules/minuscules.

La commande retourne un entier long indiquant le type de valeur. Vous pouvez comparer cette valeur aux constantes suivantes du thème **Types champs et variables** :

Constante	Type	Valeur
Est un booléen	Entier long	6
Est un null JSON	Entier long	255
Est un numérique	Entier long	1
Est un objet	Entier long	38
Est un tableau objet	Entier long	39
Est un texte	Entier long	2
Est une variable indéfinie	Entier long	5

Exemple

On souhaite obtenir le type de valeurs standard :

```
C_OBJET($ref)
OB FIXER($ref;"nom";"smith";"age";42)
$type:=OB Lire type($ref;"nom") //$type retourne 2
$type2:=OB Lire type($ref;"age") //$type2 retourne 1
```

OB SUPPRIMER

OB SUPPRIMER (objet ; propriété)

Paramètre	Type		Description
objet	Objet, Champ objet	➡	Objet structuré
propriété	Texte	➡	Nom de la propriété à supprimer

Description

La commande **OB SUPPRIMER** permet de supprimer la *propriété* de l'objet de langage désigné par le paramètre *objet*. Cette commande supprime la *propriété* ainsi que sa valeur courante.

objet doit avoir été défini via la commande **C_OBJET** ou désigner un champ objet 4D.

Passez dans le paramètre *propriété* le libellé de la propriété à lire. Attention, le paramètre *propriété* tient compte des majuscules/minuscules.

Exemple

Vous souhaitez supprimer la propriété "age" d'un objet :

```
C_OBJET($Objet)
OB FIXER($Objet;"nom";"smith";"age";42;"client";Vrai)
    //$Objet={"nom":"smith","age":42,"client":true}
OB SUPPRIMER($Objet;"age")
    //$Objet={"nom":"smith","client":true}
```

Opérateurs

Opérateurs

Les opérateurs sont des symboles permettant d'effectuer des opérations sur des expressions. Les opérateurs peuvent effectuer des calculs sur des nombres, des dates et des heures. Ils effectuent aussi des opérations logiques sur les chaînes, les booléens et des expressions logiques ainsi que des opérations spéciales sur des images. Ils combinent des expressions simples pour générer de nouvelles expressions.

Priorité

L'ordre dans lequel une expression est évaluée s'appelle la priorité. 4D applique strictement une règle de priorité de gauche à droite. **L'ordre algébrique n'est pas appliqué**. Par exemple :

```
3+4*5
```

retourne 35 car l'expression est évaluée comme 3 + 4, qui donne 7, multiplié par 5, ce qui donne 35.

Les parenthèses doivent être utilisées pour forcer l'ordre de calcul en fonction de vos besoins. Par exemple :

```
3+(4*5)
```

retourne 23 car l'expression (4 * 5) est évaluée en premier lieu. Le résultat (20) est alors ajouté à 3, ce qui donne le résultat final 23.

Des parenthèses peuvent être incluses dans d'autres parenthèses. Assurez-vous qu'il y ait une parenthèse fermante pour chaque parenthèse ouverte. Une parenthèse manquante ou placée à un mauvais endroit peut soit donner un résultat erroné, soit renvoyer une expression invalide. De plus, si vous avez l'intention de compiler vos applications, vous devez vous assurer d'une bonne utilisation des parenthèses. Le compilateur interprètera toute parenthèse manquante ou superflue comme une erreur de syntaxe.

L'opérateur d'affectation (ou d'assignation)

L'opérateur d'affectation := se distingue des autres opérateurs. Au lieu de combiner des expressions en une seule, l'opérateur d'affectation copie la valeur de l'expression située à sa droite dans la variable ou le champ qui se trouve à sa gauche. Par exemple, la ligne suivante place la valeur 4 (le nombre de caractères présents dans le mot Pont) dans la variable *maVar*, qui prend alors le type numérique.

```
maVar:=Longueur("Pont")
```

Important : Ne confondez pas l'opérateur d'affectation := avec l'opérateur de comparaison d'égalité =.

Les autres opérateurs proposés par le langage de 4D sont décrits dans les sections suivantes :

Opérateurs sur les chaînes

Référez-vous à la section **Opérateurs sur les chaînes**.

Opérateurs numériques

Référez-vous à la section **Opérateurs numériques**.

Opérateurs sur les dates

Référez-vous à la section **Opérateurs sur les dates**.

Opérateurs sur les heures

Référez-vous à la section **Opérateurs sur les heures**.

Comparateurs

Référez-vous à la section **Opérateurs de comparaison**.

Opérateurs logiques

Référez-vous à la section **Opérateurs logiques**.

Opérateurs sur les images

Référez-vous à la section **Opérateurs sur les images**.

Opérateurs sur les bits

Référez-vous à la section **Opérateurs sur les bits**.

Opérateurs sur les bits

Les opérateurs sur les bits s'appliquent à des expressions ou valeurs de type *Entier long*.

Note : Si vous passez une valeur de type *Entier* ou *Réel* à un opérateur sur les bits, 4D la convertit en *Entier long* avant de calculer le résultat de l'expression.

Lorsque vous employez des opérateurs sur les bits, vous devez considérer une valeur de type *Entier long* comme un tableau de 32 bits. Les bits sont numérotés de 0 à 31, de droite à gauche.

Comme un bit peut valoir 0 (zéro) ou 1, vous pouvez également considérer une valeur de type *Entier long* comme une expression dans laquelle vous pouvez stocker 32 valeurs de type *Booléen*. Lorsque le bit vaut 1, la valeur est **Vrai** et lorsque le bit vaut 0, la valeur est **Faux**.

Une expression utilisant un opérateur sur les bits retourne une valeur de type *Entier long*, à l'exception de l'opérateur **Tester bit** avec lequel l'expression retournée est du type *Booléen*. Le tableau suivant fournit la liste des opérateurs sur les bits et leur syntaxe :

Opération	Opérateur	Syntaxe	Retourne	
ET	&	E.long & E.long	E.long	
OU (inclusif)	|	E.long | E.long	E.long	
OU (exclusif)	^|	E.long ^| E.long	E.long	
Décaler bits à gauche	<<	E.long << E.long	E.long	(voir note 1)
Décaler bits à droite	>>	E.long >> E.long	E.long	(voir note 1)
Mettre bit à 1	?+	E.long ?+ E.long	E.long	(voir note 2)
Mettre bit à 0	?-	E.long ?- E.long	E.long	(voir note 2)
Tester bit	??	E.long ?? E.long	Booléen	(voir note 2)

Notes

1. Dans les opérations utilisant **Décaler bits à gauche** et **Décaler bits à droite**, le second opérande indique le nombre de décalages de bits du premier opérande à effectuer dans la valeur retournée. Par conséquent, ce second opérande doit être compris entre *0* et *31*. Notez qu'un décalage de *0* retourne une valeur inchangée et qu'un décalage de plus de 31 bits retourne 0x00000000 car tous les bits sont perdus. Si vous passez une autre valeur en tant que second opérande, le résultat sera non significatif.
2. Dans les opérations utilisant **Mettre bit à 1**, **Mettre bit à 0** et **Tester bit**, le second opérande indique le numéro du bit sur lequel agir. Par conséquent, ce second opérande doit être compris entre *0* et *31*, sinon le résultat de l'expression sera non significatif.

Le tableau suivant dresse la liste des opérateurs sur les bits et de leurs effets :

Opération sur les bits	Description
ET	Chaque bit retourné est le résultat de l'opération ET logique appliquée aux deux bits opérandes. Voici la table du ET logique : *1 & 1 --> 1* *0 & 1 --> 0* *1 & 0 --> 0* *0 & 0 --> 0* En résumé, le résultat vaut *1* si les deux bits opérandes valent *1*, dans tous les autres cas le bit résultant vaut *0*.
OU (inclusif)	Chaque bit retourné est le résultat de l'opération OU inclusif logique appliquée aux deux bits opérandes. Voici la table du OU inclusif logique : *1 | 1 --> 1* *0 | 1 --> 1* *1 | 0 --> 1* *0 | 0 --> 0* En résumé, le résultat vaut *1* si au moins un des deux bits opérandes vaut *1*, sinon le bit résultant vaut *0*.
OU (exclusif)	Chaque bit retourné est le résultat de l'opération OU exclusif logique appliquée aux deux bits opérandes. Voici la table du OU exclusif logique: *1 ^| 1 --> 0* *0 ^| 1 --> 1* *1 ^| 0 --> 1* *0 ^| 0 --> 0* Donc, le résultat vaut *1* si un seul des deux bits opérandes vaut *1* (et pas l'autre), dans tous les autres cas le bit résultant vaut *0*.
Décaler bits à gauche	La valeur retournée correspond au premier opérande dont la valeur est décalée vers la gauche du nombre de bits spécifié par le second opérande. Les bits auparavant situés à gauche sont perdus et les nouveaux bits situés à droite ont la valeur 0. **Note** : En ne tenant compte que des valeurs positives, un décalage vers la gauche de **N** bits revient à multiplier la valeur par *2^N*.
Décaler bits à droite	La valeur retournée correspond au premier opérande dont la valeur est décalée vers la droite du nombre de bits spécifié par le second opérande. Les bits auparavant situés à droite sont perdus et les nouveaux bits situés à gauche ont la valeur 0. **Note** : En ne tenant compte que des valeurs positives, un décalage vers la droite de **N** bits revient à diviser la valeur par *2^N*.
Mettre bit à 1	La valeur retournée est la valeur du premier opérande dans lequel le bit dont le numéro est spécifié par le second opérande est positionné à *1*. Les autres bits sont inchangés.
Mettre bit à 0	La valeur retournée est la valeur du premier opérande dans lequel le bit dont le numéro est spécifié par le second opérande est positionné à *0*. Les autres bits sont inchangés.
Tester bit	Retourne Vrai si, dans le premier opérande, le bit dont le numéro est indiqué par le second opérande vaut *1*. Retourne Faux si, dans le premier opérande, le bit dont le numéro est indiqué par le second opérande vaut *0*.

Exemple

(1) Le tableau ci-dessous propose un exemple d'utilisation de chaque opérateur sur les bits :

Opération	Exemple	Résultat
ET	0x0000FFFF & 0xFF00FF00	0x0000FF00
OU (inclusif)	0x0000FFFF | 0xFF00FF00	0xFF00FFFF
OU (exclusif)	0x0000FFFF ^| 0xFF00FF00	0xFF0000FF
Décaler bit gauche	0x0000FFFF << 8	0x00FFFF00
Décaler bit droit	0x0000FFFF >> 8	0x000000FF
Mettre bit à 1	0x00000000 ?+ 16	0x00010000
Mettre bit à 0	0x00010000 ?- 16	0x00000000
Tester bit	0x00010000 ?? 16	Vrai

(2) 4D exploite de nombreuses constantes prédéfinies. Le nom de certaines d'entre elles commence par "Bit" ou "Masque". C'est le cas des constantes incluses dans le thème **Propriétés des ressources** :

Constante	Type	Valeur
Bit ressource heap système	Entier long	6
Bit ressource modifiée	Entier long	1
Bit ressource préchargée	Entier long	2
Bit ressource protégée	Entier long	3
Bit ressource purgeable	Entier long	5
Bit ressource verrouillée	Entier long	4
Masque ressource heap système	Entier long	64
Masque ressource modifiée	Entier long	2
Masque ressource préchargée	Entier long	4
Masque ressource protégée	Entier long	8
Masque ressource purgeable	Entier long	32
Masque ressource verrouillée	Entier long	16

Ces constantes vous permettent de tester la valeur retournée par la fonction **Lire proprietes ressource** ou de définir la valeur à passer à **_o_ÉCRIRE PROPRIÉTÉS RESSOURCE**. Les constantes dont le nom débute par “Bit” fournissent la position du bit que vous voulez tester, effacer ou fixer. Les constantes dont le nom débute par “Masque” sont des valeurs de type Entier long dans lesquelles seul le bit que vous voulez tester, effacer ou fixer est égal à 1.

Si, par exemple, vous devez tester si une ressource (dont les propriétés sont stockées dans la variable *$vlResAttr*) est purgeable ou non, vous pouvez écrire :

```
Si($vlResAttr ?? Bit ressource purgeable) ` Est-ce que la ressource est purgeable?
```

ou :

```
Si(($vlResAttr  & Masque ressource purgeable)#0) ` Est-ce que la ressource est purgeable?
```

A l'inverse, vous pouvez utiliser ces constantes pour définir le même bit. Vous pouvez écrire :

```
$vlResAttr:=$vlResAttr ?+Bit ressource purgeable
```

ou :

```
$vlResAttr:=$vlResAttr |Bit ressource purgeable
```

(3) Vous voulez stocker deux valeurs entières dans un *Entier long*. Vous pouvez écrire :

```
$vlLong:=($viIntA<<16)|$viIntB  ` Stocker deux Entiers dans un Entier long
$vlIntA:=$vlLong>>16 ` Extraire l'Entier stocké dans le mot haut
$viIntB:=$vlLong  & 0xFFFF ` Extraire l'entier stocké dans le mot bas
```

Note : Soyez prudent lorsque vous manipulez des **Entiers longs** ou des **Entiers** avec des expressions qui combinent des opérateurs sur les bits et des opérateurs numériques. Le bit supérieur (bit 31 pour un Entier long, bit 15 pour un Entier) détermine le signe de la valeur (positif s'il est à 0, négatif s'il est à 1). Les opérateurs numériques utilisent ce bit pour détecter le signe d'une valeur, mais les opérateurs sur les bits n'en tiennent pas compte.

Opérateurs de comparaison

Les tableaux suivants décrivent les opérateurs de comparaison. Ces opérateurs peuvent être appliqués aux expressions de type chaîne, numérique, date, heure, pointeur et image avec métadonnées (il n'est donc pas possible de les utiliser avec des expressions de type tableau ou BLOB).
Une expression qui utilise un opérateur de comparaison retourne une valeur booléenne, soit **VRAI** soit **FAUX**.

Note : Il est possible de comparer deux images à l'aide de la commande **Images egales**.

Comparaisons de chaînes

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Chaîne = Chaîne	Booléen	"abc" = "abc"	Vrai
			"abc" = "abd"	Faux
Inégalité	Chaîne # Chaîne	Booléen	"abc" # "abd"	Vrai
			"abc" # "abc"	Faux
Supérieur à	Chaîne > Chaîne	Booléen	"abd" > "abc"	Vrai
			"abc" > "abc"	Faux
Inférieur à	Chaîne < Chaîne	Booléen	"abc" < "abd"	Vrai
			"abc" < "abc"	Faux
Supérieur ou égal à	Chaîne >= Chaîne	Booléen	"abd" >= "abc"	Vrai
			"abc" >= "abd"	Faux
Inférieur ou égal à	Chaîne <= Chaîne	Booléen	"abc" <= "abd"	Vrai
			"abd" <= "abc"	Faux
Contient mot-clé	Chaîne % Chaîne	Booléen	"Alpha Bravo" % "Bravo"	Vrai
			"Alpha Bravo" % "ravo"	Faux
	Image % Chaîne	Booléen	*Expr_image* % "Mer"	Vrai (*)

(*) Si le mot-clé "Mer" a été associé à l'image stockée dans l'expression image (champ ou variable).
Important : Des informations supplémentaires sur les comparaisons de chaînes sont fournies à la fin de cette section.

Comparaisons de numériques

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Nombre = Nombre	Booléen	10 = 10	Vrai
			10 = 11	Faux
Inégalité	Nombre # Nombre	Booléen	10 #11	Vrai
			10 # 10	Faux
Supérieur à	Nombre > Nombre	Booléen	11 > 10	Vrai
			10 > 11	Faux
Inférieur à	Nombre < Nombre	Booléen	10 < 11	Vrai
			11 < 10	Faux
Supérieur ou égal à	Nombre >= Nombre	Booléen	11 >= 10	Vrai
			10 >= 11	Faux
Inférieur ou égal à	Nombre <= Nombre	Booléen	10 <= 11	Vrai
			11 <= 10	Faux

Comparaisons de dates

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Date = Date	Booléen	!1/1/97! =!1/1/97!	Vrai
			!20/1/97! =!1/1/97!	Faux
Inégalité	Date # Date	Booléen	!20/1/97! # !1/1/97!	Vrai
			!1/1/97! # !1/1/97!	Faux
Supérieur à	Date > Date	Booléen	!20/1/97! > !1/1/97!	Vrai
			!1/1/97! > !1/1/97!	Faux
Inférieur à	Date < Date	Booléen	!1/1/97! < !20/1/97!	Vrai
			!1/1/97! < !1/1/97!	Faux
Supérieur ou égal à	Date >= Date	Booléen	!20/1/97! >=!1/1/97!	Vrai
			!1/1/97!>=!20/1/97!	Faux
Inférieur ou égal à	Date <= Date	Booléen	!1/1/97!<=!20/1/97!	Vrai
			!20/1/97!<=!1/1/97!	Faux

Comparaisons d'heures

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Heure = Heure	Booléen	?01:02:03? = ?01:02:03?	Vrai
			?01:02:03? = ?01:02:04?	Faux
Inégalité	Heure # Heure	Booléen	?01:02:03? # ?01:02:04?	Vrai
			?01:02:03? # ?01:02:03?	Faux
Supérieur à	Heure > Heure	Booléen	?01:02:04? > ?01:02:03?	Vrai
			?01:02:03? > ?01:02:03?	Faux
Inférieur à	Heure < Heure	Booléen	?01:02:03? < ?01:02:04?	Vrai
			?01:02:03? < ?01:02:03?	Faux
Supérieur ou égal à	Heure >= Heure	Booléen	?01:02:03? >=?01:02:03?	Vrai
			?01:02:03? >=?01:02:04?	Faux
Inférieur ou égal à	Heure <= Heure	Booléen	?01:02:03? <=?01:02:03?	Vrai
			?01:02:04? <=?01:02:03?	Faux

Comparaisons de pointeurs

Avec :

```
 ` vPtrA et vPtrB pointent sur le même objet
vPtrA:=->unObjet
vPtrB:=->unObjet
 ` vPtrC pointe sur un autre objet
vPtrC:=->autreObjet
```

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Pointeur = Pointeur	Booléen	vPtrA = vPtrB	Vrai
			vPtrA = vPtrC	Faux
Inégalité	Pointeur # Pointeur	Booléen	vPtrA # vPtrC	Vrai
			vPtrA # vPtrB	Faux

Remarques sur les comparaisons de chaînes

Voici quelques informations supplémentaires sur les comparaisons d'alphanumériques :

- Les chaînes sont toujours comparées caractère par caractère (hormis en cas de recherche par mot-clé, cf. ci-dessous).
- Lors d'une comparaison de chaînes, 4D ne tient pas compte de la casse des caractères ; par exemple, "a"="A" retourne VRAI. Pour savoir si des caractères sont en majuscules ou en minuscules, vous devez comparer leurs codes de caractères. Par exemple, l'expression suivante retourne FAUX :

```
Code de caractere("A")=Code de caractere("a") // 65 n'est pas égal à 97
```

- Lors d'une comparaison de chaînes, les caractères diacritiques sont comparés à l'aide de la table de comparaison des caractères de votre machine. Par exemple, les expressions suivantes retournent VRAI :

```
"n"="ñ"
"n"="Ñ"
"A"="å"
 // etc
```

- A la différence des autres comparaisons de chaîne, les **recherches par mots-clés** recherchent des “mots” dans des “textes” : les mots sont évalués individuellement et dans leur globalité. L’opérateur % retournera toujours Faux si la recherche porte sur plusieurs mots ou une partie de mot (par exemple une syllabe). Les “mots” sont des chaînes de caractères encadrées par des “séparateurs”, qui sont les espaces, les caractères de ponctuation et les tirets. Une apostrophe, comme dans “aujourd'hui”, est généralement considérée comme partie du mot, mais sera ignorée dans certains cas (cf. règles ci-dessous). Les nombres peuvent être recherchés car ils sont évalués dans leur ensemble (incluant les symboles décimaux). Les autres symboles (monnaie, température, etc.) seront ignorés.

```
"Alpha Bravo Charlie"%"Bravo" ` Retourne Vrai
"Alpha Bravo Charlie"%"vo" ` Retourne Faux
"Alpha Bravo Charlie"%"Alpha Bravo" ` Retourne Faux
"Alpha,Bravo,Charlie"%"Alpha" ` Retourne Vrai
"Software and Computers"%"comput@" ` Retourne Vrai
```

Notes :
- 4D utilise la librairie ICU pour la détection des mots-clés. Pour plus d'informations sur les règles mises en oeuvre, reportez-vous à l'adresse http://www.unicode.org/unicode/reports/tr29/#Word_Boundaries.
- En version japonaise, 4D utilise par défaut la librairie *Mecab* en lieu et place de ICU pour la détection des mots-clés. Pour plus d'informations, reportez-vous au paragraphe **Prise en charge de Mecab (version japonaise)**.

- Le joker (@) peut être utilisé dans toute comparaison de chaînes. Il remplace un ou plusieurs caractères. Ainsi, par exemple, l'expression suivante est évaluée à VRAI :

```
"abcdefghij"="abc@"
```

Le joker doit être utilisé dans le second opérande (la chaîne qui se trouve à droite de l'opérateur). L'expression suivante est évaluée à FAUX car le joker est alors considéré en tant que caractère :

```
"abc@"="abcdefghij"
```

Le joker signifie “un ou plusieurs caractères sinon rien”. Les expressions suivantes sont évaluées à VRAI :

```
"abcdefghij"="abcdefghij@"
"abcdefghij"="@abcdefghij"
"abcdefghij"="abcd@efghij"
"abcdefghij"="@abcdefghij@"
"abcdefghij"="@abcde@fghij@"
```

En revanche, dans tous les cas, lorsque deux jokers consécutifs sont placés dans une comparaison de chaînes, celle-ci sera évaluée à FAUX. L'expression suivante est à FAUX :

```
"abcdefghij"="abc@@fg"
```

Lorsque l'opérateur de comparaison est ou contient un symbole < ou >, seule la comparaison avec un seul joker situé en fin d'opérande est prise en charge :

```
"abcd"<="abc@" `Comparaison valide
"abcd"<="abc@ef" `Comparaison non valide
```

Note : Si vous souhaitez effectuer des comparaisons ou des recherches utilisant @ en tant que caractère (et non en tant que joker), vous disposez de deux possibilités :

- Utiliser l'instruction **Code de caractere**(Arobase).
  Imaginons par exemple que vous souhaitiez savoir si une chaîne se termine par le caractère @.
  - l’expression suivante (si *$vaValeur* n'est pas vide) retourne toujours VRAI :

```
($vaValeur≤Longueur($vaValeur)≥="@")
```

  - l'expression suivante sera correctement évaluée :

```
(Code de caractere($vaValeur≤Longueur($vaValeur)≥)#64)
```

- Utiliser l'option “Considérer @ comme joker uniquement au début et à la fin des chaînes de caractères”, accessible via la boîte de dialogue des Propriétés de la base.
  Cette option permet de paramétrer le mode d’interprétation du caractère @ lorsque celui-ci est inclus dans une chaîne de caractères. Elle peut donc influer sur le fonctionnement des opérateurs de comparaison utilisés dans le cadre de recherches ou de tris. Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.

Opérateurs sur les dates

Une expression qui utilise un opérateur sur les dates retourne une date ou une valeur numérique, suivant l'opération effectuée. Toutes les opérations sur les dates retournent des valeurs exactes, tenant compte en particulier des années bissextiles. Le tableau suivant décrit les opérateurs sur les dates :

Opération	Syntaxe	Retourne	Expression	Valeur
Différence	Date - Date	Nombre	!1997-01-20! - !1997-01-01!	19
Addition	Date + Nombre	Date	!1997-01-20! + 9	!1997-01-29!
Soustraction	Date - Nombre	Date	!1997-01-20! - 9	!1997-01-11!

Opérateurs logiques

4D supporte deux opérateurs logiques : l'opérateur d'intersection (ET) et l'opérateur de réunion inclusive (OU). Ces deux opérateurs ne s'appliquent qu'aux expressions booléennes. Le ET logique retourne VRAI si les deux expressions sont VRAIES. Le OU logique retourne VRAI si au moins une des expressions est VRAIE.

4D vous permet également d'exploiter les fonctions booléennes **Vrai**, **Faux** et **Non**. Pour plus d'informations, reportez-vous aux descriptions de ces commandes.

Le tableau suivant décrit les opérateurs logiques :

Opération	Syntaxe	Retourne	Expression	Valeur
ET	Booléen & Booléen	Booléen	("A" = "A") & (15 # 3)	Vrai
			("A" = "B") & (15 # 3)	Faux
			("A" = "B") & (15 = 3)	Faux
OU	Booléen | Booléen	Booléen	("A" = "A") | (15 # 3)	Vrai
			("A" = "B") | (15 # 3)	Vrai
			("A" = "B") | (15 = 3)	Faux

Voici la "table de vérité" pour l'opérateur logique "ET" :

Expr1	Expr2	Expr1 & Expr2
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

Voici la "table de vérité" pour l'opérateur logique "OU" :

Expr1	Expr2	Expr1 | Expr2
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Astuce : Si vous devez calculer une réunion exclusive (le "OU" exclusif) entre Expr1 et Expr2, écrivez :

```
(Expr1|Expr2) & Non(Expr1 & Expr2)
```

Opérateurs numériques

Une expression qui utilise un opérateur numérique retourne une valeur numérique. Le tableau suivant décrit les opérateurs numériques :

Opération	Syntaxe	Retourne	Expression	Valeur
Addition	Nombre + Nombre	Nombre	2 + 3	5
Soustraction	Nombre – Nombre	Nombre	3 – 2	1
Multiplication	Nombre * Nombre	Nombre	5 * 2	10
Division	Nombre /Nombre	Nombre	5 / 2	2.5
Division entière	Nombre \ Nombre	Nombre	5 \ 2	2
Modulo	Nombre % Nombre	Nombre	5 % 2	1
Exponentiation	Nombre ^ Nombre	Nombre	2 ^ 3	8

L'opérateur modulo % divise le premier nombre par le second et retourne le reste de la division entière. Voici quelques exemples :

- 10 % 2 retourne 0 car la division de 10 par 2 ne donne pas de reste.
- 10 % 3 retourne 1 car le reste est 1.
- 10,5 % 2 retourne 0 car le reste n'est pas un nombre entier.

ATTENTION

- L'opérateur modulo % retourne des valeurs significatives avec des nombres appartenant à la catégorie des entiers longs (de –2^31 à +2^31 moins 1). Pour calculer le modulo de nombres qui ne sont pas dans cet intervalle, utilisez la fonction **Modulo**.
- L'opérateur division entière \ retourne des valeurs significatives avec des nombres entiers uniquement.

Opérateurs sur les images

Le tableau suivant décrit les opérateurs que vous pouvez utiliser avec 4D sur les images. Une expression qui utilise un opérateur sur les images retourne toujours une image.

Opération	Syntaxe	Action
Concaténation horizontale	Image1 + Image2	Place Image2 à la droite d'Image1
Concaténation verticale	Image1 / Image2	Place Image2 au-dessous d'Image1
Superposition exclusive(*)	Image1 & Image2	Superpose Image2 à Image1 (Image2 est au premier plan)
Superposition inclusive(*)	Image1 | Image2	Superpose Image2 à Image1 et retourne le masque résultant si les deux images sont de même taille
Déplacement horizontal	Image + Nombre	Déplace Image horizontalement d'un nombre de pixels égal à Nombre
Déplacement vertical	Image / Nombre	Déplace Image verticalement d'un nombre de pixels égal à Nombre
Redimensionnement	Image * Nombre	Redimensionne Image au pourcentage Nombre
Extension horizontale	Image *+ Nombre	Redimensionne Image horizontalement au pourcentage Nombre
Extension verticale	Image */ Nombre	Redimensionne Image verticalement au pourcentage Nombre

(*) Le fonctionnement des opérateurs de superposition exclusive (&) et superposition inclusive (|) a été modifié à compter de 4Dv14 suite à la mise à jour des librairies de gestion d'affichage utilisées par le programme.
Image3 := Image1 & Image2 produit le même résultat que:

```
COMBINER IMAGES(Image3;Image1;Superposition;Image2)
```

Image3 := Image1 | Image2 produit le même résultat que:

```
$egal:=Images egales(Image1;Image2;Image3)
```

A noter que pour que l'opérateur **|** puisse être utilisé, Image1 et Image2 doivent être strictement de la même dimension. Si les deux images sont de taille différente, l'opération Image1 | Image2 produit une image vide.

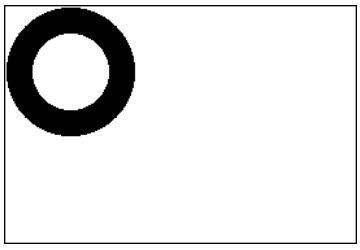
Note : La commande **COMBINER IMAGES** permet d'effectuer des superpositions en conservant les caractéristiques de chaque image source dans l'image résultante.

Les opérateurs sur les images retournent des images vectorielles si les deux images source sont elles aussi vectorielles (rappelez-vous qu'une image imprimée avec le format d'affichage Sur fond est imprimée en tant que bitmap).

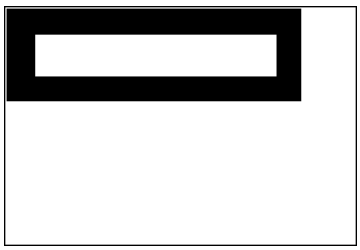
Exemple

Toutes les images qui sont affichées utilisent le format d'affichage **Image sur fond**.

Voici l'image *cercle* :



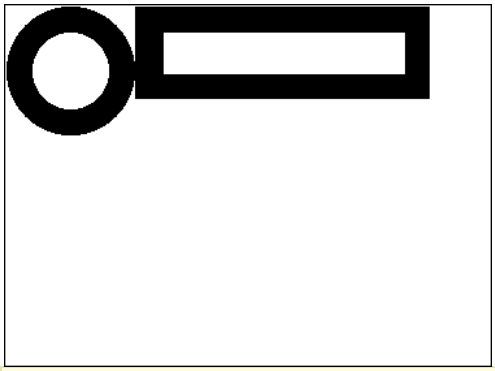
Voici l'image *rectangle* :



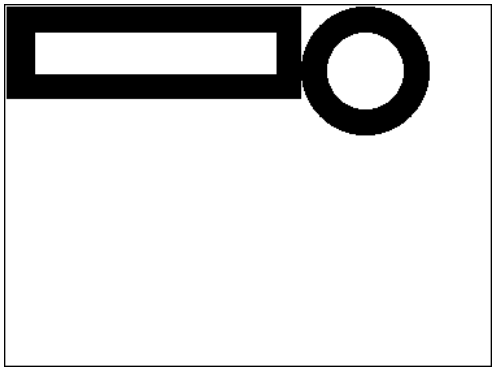
Dans les exemples ci-dessous, chaque expression est suivie de sa représentation graphique.

- Concaténation horizontale

```
cercle+rectangle ` Placer le rectangle à droite du cercle
```

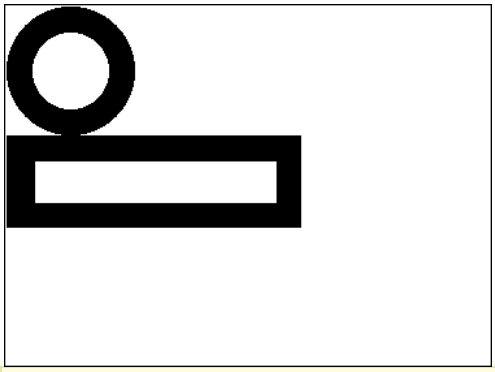


```
rectangle+cercle ` Placer le cercle à droite du rectangle
```

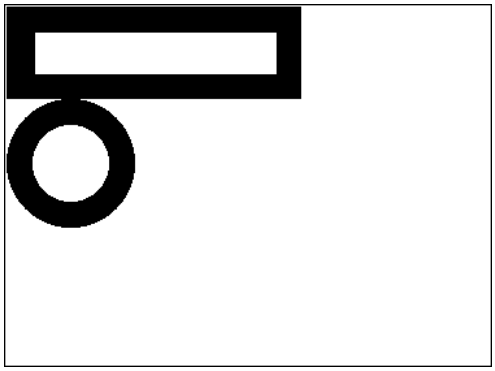


- Concaténation verticale

```
cercle/rectangle ` Placer le rectangle sous cercle
```

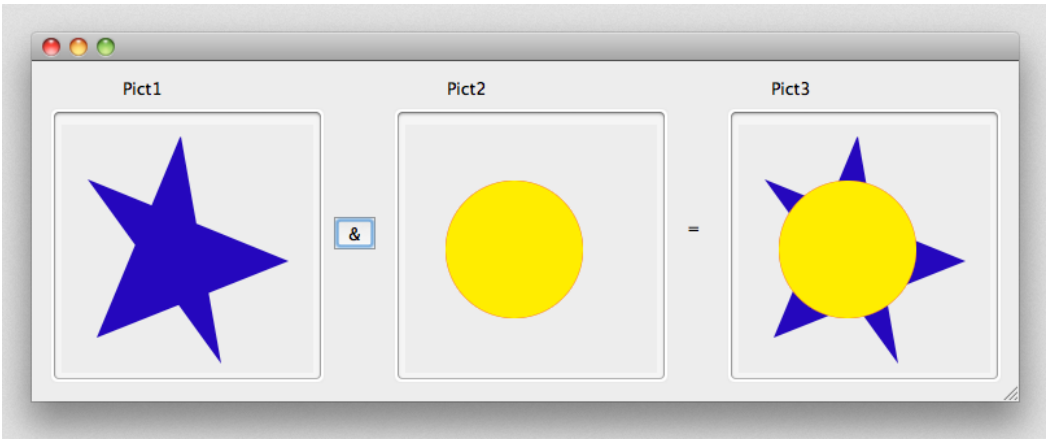


```
rectangle/cercle ` Placer le cercle sous le rectangle
```



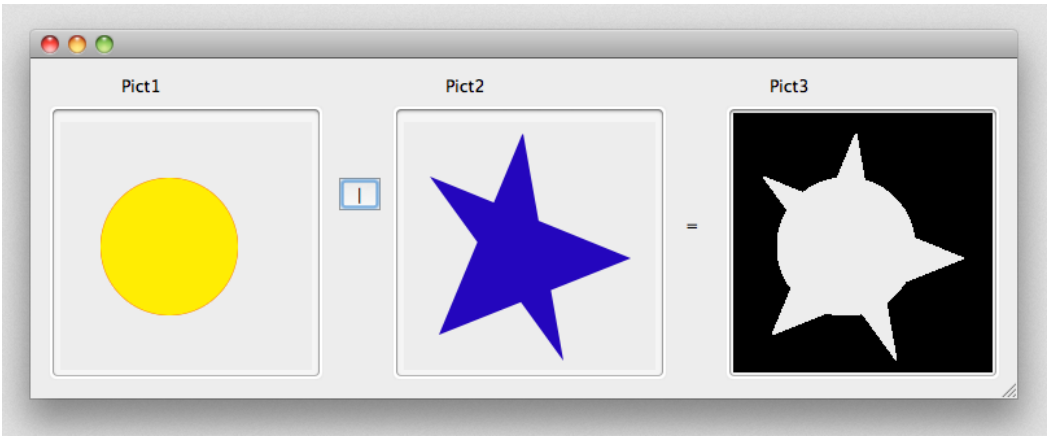
- Superposition exclusive

```
Pict3:=Pict1 & Pict2 // Superposer Pict2 à Pict1
```



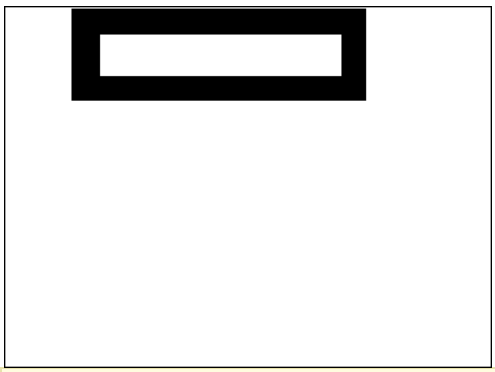
- Superposition inclusive

```
Pict3:=Pict1|Pict2 // Récupérer le masque résultant de la superposition de deux images de même taille
```

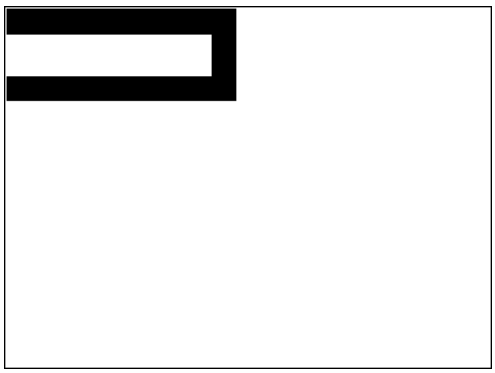


- Déplacement horizontal

```
rectangle+50 ` Déplacer le rectangle 50 pixels vers la droite
```

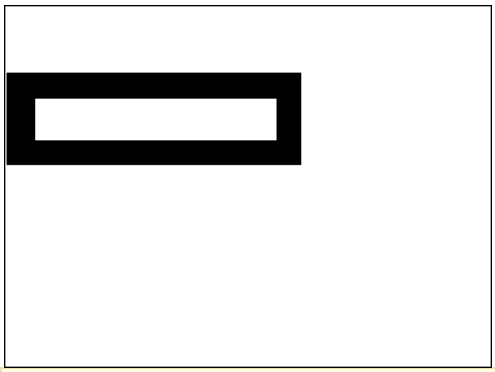


```
rectangle-50 ` Déplacer le rectangle 50 pixels vers la gauche
```

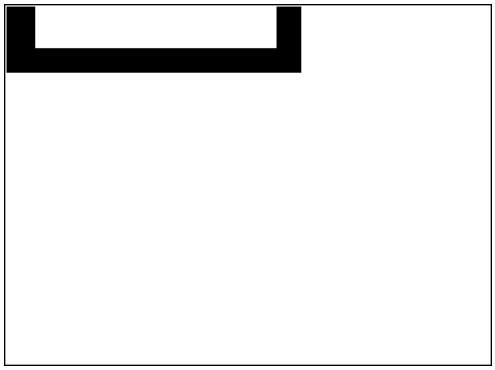


- Déplacement vertical

```
rectangle/50 ` Déplacer le rectangle 50 pixels vers le bas
```



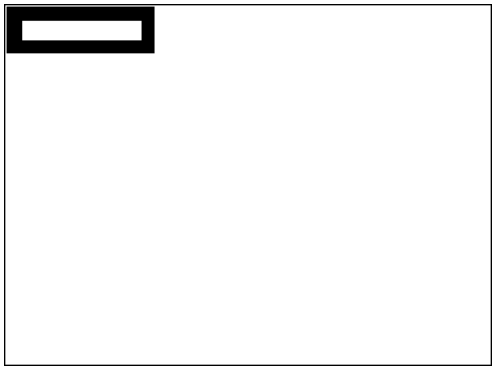
```
rectangle/-20 ` Déplacer le rectangle 20 pixels vers le haut
```



- Redimensionnement

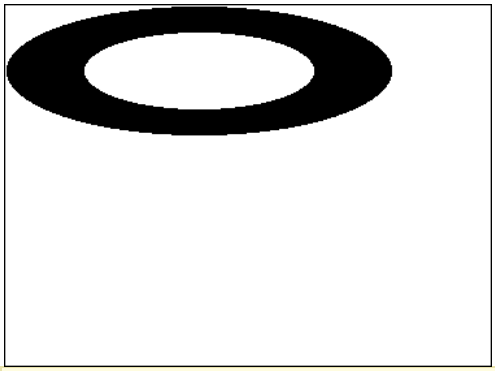
```
rectangle*1.5 ` Augmenter la taille du rectangle de 50%
```

```
rectangle*0.5 ` Réduire la taille du rectangle de 50%
```



- Extension horizontale

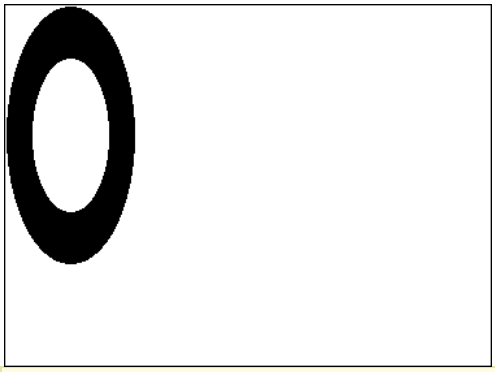
```
cercle*+3 `Multiplier par 3 la largeur du cercle
```



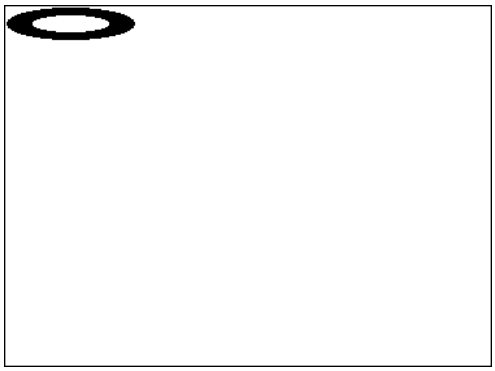
```
cercle*+0,25 ` La largeur du cercle est réduite à un quart de sa taille originale
```

- Extension verticale

```
cercle*/2 ` Doubler la hauteur du cercle
```



```
cercle*/0.25 ` La hauteur du cercle est réduite à un quart de sa taille originale
```



Opérateurs sur les chaînes

Une expression qui utilise un opérateur sur les chaînes alphanumériques retourne une chaîne. Le tableau suivant décrit les opérateurs sur les chaînes :

Opération	Syntaxe	Retourne	Expression	Valeur
Concaténation	Chaîne + Chaîne	Chaîne	"abc" + "def"	"abcdef"
Répétition	Chaîne * Nombre	Chaîne	"ab" * 3	"ababab"

Opérateurs sur les heures

Une expression qui utilise un opérateur sur les heures retourne une heure ou une valeur numérique, suivant l'opération effectuée. Le tableau suivant décrit les opérateurs sur les heures :

Opération	Syntaxe	Retourne	Expression	Valeur
Addition	Heure + Heure	Heure	?02:03:04? + ?01:02:03?	?03:05:07?
Soustraction	Heure - Heure	Heure	?02:03:04? - ?01:02:03?	?01:01:01?
Addition	Heure + Nombre	Nombre	?02:03:04? + 65	7449
Soustraction	Heure - Nombre	Nombre	?02:03:04? - 65	7319
Multiplication	Heure * Nombre	Nombre	?02:03:04? * 2	14768
Division	Heure / Nombre	Nombre	?02:03:04? / 2	3692
Division entière	Heure \ Nombre	Nombre	?02:03:04? \ 2	3692
Modulo	Heure % Heure	Heure	?20:10:00? % ?04:20:00?	?02:50:00?
Modulo	Heure % Nombre	Nombre	?02:03:04? % 2	0

Exemple 1

Vous pouvez combiner des expressions de type heure et de type numérique à l'aide des fonctions **Heure** ou **Heure courante**. Par exemple :

```
  // La ligne suivante assigne à la variable $vlSecondes le nombre de secondes qui, dans une heure à partir de
maintenant, se seront écoulées depuis minuit
 $vlSecondes:=Heure courante+3600
  // La ligne suivante assigne à la variable $vhBientôt l'heure qu'il sera dans une heure
 $vhBientôt:=Heure(Heure courante+3600)
```

La seconde ligne peut également être écrite de la façon suivante :

```
  // La ligne suivante assigne à la variable $vhBientôt l'heure qu'il sera dans une heure
 $vhBientôt:=Heure courante+?01:00:00?
```

Exemple 2

Il faut parfois convertir une expression heure en expression numérique. Par exemple, vous ouvrez un document sur disque à l'aide de la fonction **Ouvrir document**, qui retourne un numéro de référence de document (**DocRef**) qui est une expression de type heure. Vous pouvez passer **DocRef** à une routine de plug-in 4D qui attend une valeur numérique comme numéro de référence de document. Dans ce cas, ajoutez *0 (*zéro) à l'heure pour obtenir une valeur numérique, sans la modifier. Par exemple :

```
  ` Sélectionner et ouvrir un document
 $vhDocRef:=Ouvrir document("")
 Si(OK=1)
  ` Passez l'expression heure DocRef en tant qu'expression numérique à une routine d'extension 4D
   faire quelque chose(0+$vhDocRef)
 Fin de si
```

Exemple 3

L'opérateur Modulo permet notamment d'ajouter des heures en tenant compte du format sur 24 heures d'une journée :

```
$t1:=?23:00:00? // il est 23h
    //on souhaite ajouter 2 heures 30
$t2:=$t1 +?02:30:00? // avec une addition simple, $t2 vaut ?25:30:00?
$t2:=($t1 +?02:30:00?)%?24:00:00? // $t2 vaut ?01:30:00?, il est bien 1h30 le lendemain
```

Outils

- Choisir
- DECODER BASE64
- ENCODER BASE64
- FIXER PARAMETRE MACRO
- FIXER VARIABLE ENVIRONNEMENT
- Generer digest
- Generer UUID
- LANCER PROCESS EXTERNE
- LIRE APERCU ACTIVITE
- LIRE PARAMETRE MACRO
- OUVRIR URL
- TRAITER BALISES 4D

Choisir

Choisir (critère ; valeur {; valeur2 ; ... ; valeurN}) -> Résultat

Paramètre	Type		Description
critère	Booléen, Entier	➡	Valeur à tester
valeur	Expression	➡	Valeurs possibles
Résultat	Expression	⮌	Valeur de critère

Description

La commande **Choisir** retourne l'une des valeurs passées dans les paramètres *valeur*, *valeur2*, etc. en fonction de la valeur du paramètre *critère*.

Vous pouvez passer un paramètre *critère* de type booléen ou numérique :

- Si *critère* est un booléen, **Choisir** retourne *valeur* si le booléen vaut Vrai et *valeur2* si le booléen vaut Faux. Dans ce cas, la commande attend exactement trois paramètres : *critère*, *valeur* et *valeur2*.
- Si *critère* est un entier, **Choisir** retourne la valeur dont la position correspond à *critère*. Attention, la numérotation des valeurs débute à 0 (la position de *valeur* est 0). Dans ce cas, la commande attend au minimum deux paramètres : *critère* et *valeur*.

La commande accepte tous les types de données pour le(s) paramètre(s) *valeur*, hormis les images, pointeurs, BLOBS et tableaux. Veillez cependant à ce que toutes les valeurs passées soient du même type, 4D n'effectue pas de vérification sur ce point.

Si aucune *valeur* ne correspond à *critère*, **Choisir** retourne une valeur "nulle" en rapport avec le type du paramètre *valeur* (par exemple 0 pour le type numérique, "" pour le type chaîne, etc.).

Cette commande permet de générer du code concis en remplacement des tests du type "Au cas ou" sur plusieurs lignes (cf. exemple 2). Elle est également très utile dans les emplacements où des formules peuvent être exécutées : éditeur de recherches, appliquer une formule, éditeur d'états rapides, colonne calculée de list box, etc.

Exemple 1

Voici une utilisation type de la commande avec un critère booléen :

```
vTitre:=Choisir([Personne]Masculin;"Mr";"Madame")
```

Ce code est strictement équivalent à :

```
Si([Personne]Masculin)
   vTitre:="Mr"
Sinon
   vTitre:="Madame"
Fin de si
```

Exemple 2

Voici une utilisation type de la commande avec un critère numérique :

```
vStatut:=Choisir([Personne]Statut;"Célibataire";"Marié";"Veuf";"Divorcé")
```

Ce code est strictement équivalent à :

```
Au cas ou
   :([Personne]Statut=0)
      vStatut:="Célibataire"
   :([Personne]Statut=1)
      vStatut:="Marié"
   :([Personne]Statut=2)
      vStatut:="Veuf"
   :([Personne]Statut=3)
      vStatut:="Divorcé"
Fin de cas
```

DECODER BASE64

DECODER BASE64 ({texteEncodé ;} blob)

Paramètre	Type		Description
texteEncodé	Texte	⇒	Texte contenant un BLOB encodé en base64
blob	BLOB	⇒	BLOB encodé en base64 (si texteEncodé omis)
		⇐	BLOB décodé

Description

La commande **DECODER BASE64** permet de décoder le BLOB encodé en base64 passé dans le paramètre *texteEncodé* ou *blob*.

Si vous passez le paramètre *texteEncodé*, la commande décode son contenu et le retourne dans le paramètre *blob*. Il doit contenir un BLOB encodé en base64. Dans ce cas, le contenu initial du paramètre *blob* est ignoré par la commande.
Si vous omettez le paramètre *texteEncodé*, la commande modifie directement le BLOB passé dans le paramètre *blob*.

La commande n'effectue pas de contrôle sur le contenu du paramètre *texteEncodé* ou *blob*. Vous devez veiller à ce que les données passées soient effectivement encodées en base64, sinon le résultat obtenu ne sera pas correct.

Exemple

Cet exemple permet de transférer une image via un BLOB :

```
C_BLOB($blobSource)
C_IMAGE($monimage)
$monimage:=[personnes]photo
IMAGE VERS BLOB($monimage;$blobSource;".JPG")
C_TEXTE($texteBASE64)
ENCODER BASE64($blobSource;$texteBASE64) //Encodage du texte
 // le binaire est maintenant disponible sous forme de chaîne de caractères dans $texteBASE64

C_TEXTE($texteBASE64)
C_BLOB($blobCible)
DECODER BASE64($texteBASE64;$blobCible) //Décodage du texte
 // le binaire encodé en base 64 est maintenant disponible sous forme de BLOB dans $blobCible
```

ENCODER BASE64

ENCODER BASE64 (blob {; texteEncodé})

Paramètre	Type		Description
blob	BLOB	⇨	BLOB à encoder en base64
		⇦	BLOB encodé en base64 (si texteEncodé omis)
texteEncodé	Texte	⇦	Résultat de blob encodé en base64

Description

La commande **ENCODER BASE64** encode le BLOB passé dans le paramètre *blob* en base64.

Si vous passez le paramètre *texteEncodé*, il reçoit sous forme de texte le contenu encodé de *blob* à l'issue de l'exécution de la commande. Dans ce cas, le paramètre *blob* lui-même n'est pas modifié par la commande.
Si vous omettez le paramètre *texteEncodé*, la commande modifie directement le BLOB passé en paramètre.

L'encodage base64 modifie des données codées sur 8 bits afin qu'elles ne conservent plus que 7 bits utiles. Cet encodage est par exemple requis pour la manipulation des BLOBs via le XML.

FIXER PARAMETRE MACRO

FIXER PARAMETRE MACRO (sélecteur ; paramTexte)

Paramètre	Type		Description
sélecteur	Entier long	⇒	Sélection à utiliser
paramTexte	Texte	⇒	Texte envoyé

Description

La commande **FIXER PARAMETRE MACRO** insère le texte *paramTexte* dans la méthode depuis laquelle elle a été appelée.

Si du texte était sélectionné dans la méthode, le paramètre *sélecteur* permet de définir si le texte *paramTexte* doit remplacer la totalité de la méthode ou uniquement le texte sélectionné. Vous pouvez passer dans *sélecteur* l'une des constantes suivantes, placées dans le thème "**Environnement 4D**" :

Constante	Type	Valeur
Texte méthode	Entier long	1
Texte méthode surligné	Entier long	2

Si aucun texte n'était sélectionné, *paramTexte* est inséré dans la méthode.

Note

Pour que les commandes **LIRE PARAMETRE MACRO** et **FIXER PARAMETRE MACRO** fonctionnent correctement, l'attribut "version" doit être déclaré dans la macro elle-même, de la façon suivante :

```
<macro name="MaMacro" version="2">
--- Texte de la macro ---
</macro>
```

Exemple

Cette macro construit un nouveau texte qui sera retourné à la méthode appelante

```
C_TEXTE($texte_entrée)
C_TEXTE($texte_sortie)
LIRE PARAMETRE MACRO(Texte méthode surligné;$texte_entrée)
 `Supposons que le texte sélectionné est une table, i.e. "[Clients]"
$texte_sortie:=""
$texte_sortie:=$texte_sortie+Nom commande(47)+"("+$texte_entrée+")" ` Tout sélectionner ([Clients])
$texte_sortie:=$texte_sortie+"$i:="+Nom commande(76)+"("+$texte_entrée+")" ` $i:=Enregistrements
trouves([Clients])
FIXER PARAMETRE MACRO(Texte méthode surligné;$texte_sortie)
 `On remplace le texte sélectionné par le nouveau code
```

FIXER VARIABLE ENVIRONNEMENT

FIXER VARIABLE ENVIRONNEMENT (nomVar ; valeurVar)

Paramètre	Type		Description
nomVar	Chaîne	⇒	Nom de la variable à fixer
valeurVar	Chaîne	⇒	Valeur de la variable ou "" pour rétablir la valeur par défaut

Description

La commande **FIXER VARIABLE ENVIRONNEMENT** vous permet de fixer la valeur d'une variable d'environnement sous Mac OS X et Windows. Elle est destinée à une utilisation conjointe avec la commande **LANCER PROCESS EXTERNE**. Elle fonctionne également avec la commande **PHP Executer**.

Passez dans le paramètre *nomVar* le nom de la variable à définir et dans le paramètre *valeurVar* sa valeur.

- Pour obtenir la liste générale des variables d'environnement et leurs valeurs possibles, reportez-vous à la documentation technique de votre système d'exploitation.
- Pour connaître la liste des variables d'environnement utilisables avec la commande **LANCER PROCESS EXTERNE**, reportez-vous à la documentation de cette commande. A noter que trois variables d'environnement spécifiques sont disponibles dans le cadre de l'utilisation dans ce contexte :
  _4D_OPTION_CURRENT_DIRECTORY : permet de définir le répertoire courant du process externe à lancer. Vous devez passer dans *valeurVar* le chemin d'accès du répertoire (syntaxe type HFS sous Mac OS et DOS sous Windows).
  _4D_OPTION_HIDE_CONSOLE (Windows uniquement) : permet de masquer la fenêtre de la console DOS. Vous devez passer "true" dans *valeurVar* pour masquer la console ou "false" pour l'afficher.
  _4D_OPTION_BLOCKING_EXTERNAL_PROCESS : permet d'exécuter le process externe en mode asynchrone, c'est-à-dire non bloquant pour les autres applications. Vous devez passer "false" dans *valeurVar* pour définir une exécution asynchrone ou "true" pour une exécution synchrone (il n'est pas possible d'utiliser une valeur par défaut avec cette variable).
  Ces variables sont valides dans le process courant pour le prochain appel à **LANCER PROCESS EXTERNE**.

Exemple

Reportez-vous aux exemples de la commande **LANCER PROCESS EXTERNE**.

Generer digest

Generer digest (param ; algorithme) -> Résultat

Paramètre	Type		Description
param	BLOB, Variable texte	➡	Blob ou texte pour lequel obtenir une clé digest
algorithme	Entier long	➡	Algorithme utilisé pour retourner la clé : 0 = Digest MD5, 1 = Digest SHA1, 2 = Digest 4D
Résultat	Texte	⬅	Valeur de la clé digest

Description

La commande **Generer digest** retourne la clé digest d'un BLOB ou d'un texte après application d'un algorithme de cryptage.

Dans 4D, les algorithmes suivants sont disponibles : **MD5** (*Message Digest 5*) et **SHA-1** (*Secure Hash 1*) ainsi que **4D** (algorithme interne). Ces algorithmes sont des fonctions de hachage différentes :

- MD5 est une séquence de 16 octets retournée en tant que chaîne de 32 caractères hexadécimaux.
- SHA-1 est une séquence de 20 octets retournée en tant que chaîne de 40 caractères hexadécimaux.
- 4D désigne l'algorithme interne utilisé par 4D pour crypter les mots de passe des utilisateurs. L'utilisation de cet algorithme est particulièrement utile dans le cadre de la **Méthode base Sur authentification 4D Mobile** lorsque vous souhaitez exploiter votre propre liste d'utilisateurs.

La valeur retournée pour un même objet sera identique sur toutes les plates-formes (Mac/Windows, 32 ou 64 bits). Le calcul est effectué à partir de la représentation en UTF8 du texte passé en paramètre.

Note : Si vous utilisez la commande avec un texte/BLOB vide, elle ne retournera pas *void* mais la valeur suivante : "d41d8cd98f00b204e9800998ecf8427e" (MD5) ou "da39a3ee5e6b4b0d3255bfef95601890afd80709" (SHA-1).

Passez un champ ou une variable Texte ou BLOB dans le paramètre *param*. La clé digest est retournée sous forme de chaîne par la fonction **Generer digest**.

Passez dans le paramètre *algorithme* une valeur désignant la fonction de hachage à employer. Vous pouvez utiliser l'une des constantes suivantes, placées dans le thème **Type digest** :

Constante	Type	Valeur	Comment
Digest 4D	Entier long	2	Utiliser l'algorithme interne de 4D
Digest MD5	Entier long	0	Utiliser l'algorithme MD5
Digest SHA1	Entier long	1	Utiliser l'algorithme SHA-1

Si le calcul de la clé digest ne s'exécute pas correctement, la fonction génère une erreur que vous pouvez intercepter à l'aide de la commande **APPELER SUR ERREUR**, et la fonction retourne une chaîne vide.

Exemple 1

Cet exemple vous permet de comparer deux documents à l'aide de l'algorithme MD5 :

```
PROPRIETES PLATE FORME($Platf;$Syst;$vlMachine)
 // Ouvrir le premier document en lecture seule
$Same:=Vrai
$vhDocRef1:=Ouvrir document("";"*";Mode lecture))
Si(OK=1)  // Si un document a été sélectionné
   DOCUMENT VERS BLOB(Document;$FirstBlob)  // Charger le document
   Si(OK=1)
      Si($Platf=Mac OS)
         DOCUMENT VERS BLOB(Document;$FirstBlobRF;*)
 // Sous Mac OS, charger la resource fork
         $MD5_1RF:=Generer digest($FirstBlobRF;Digest MD5)
      Fin de si

 // Ouvrir le second document en lecture seule
      $vhDocRef2:=Ouvrir document("";"*";Mode lecture))
      Si(OK=1)
         DOCUMENT VERS BLOB(Document;$SecondBlob)
         Si(OK=1)
            Si($Platf=Mac OS)
               DOCUMENT VERS BLOB(Document;$SecondBlobRF;*)
               $MD5_2RF:=Generer digest($SecondBlobRF;Digest MD5)
               Si($MD5_1RF#$MD5_2RF) // Comparer les digests
                  $Same:=Faux
               Fin de si
            Fin de si
            $MD5_1:=Generer digest($FirstBlob;Digest MD5)
            $MD5_2:=Generer digest($SecondBlob;Digest MD5)
            Si(($MD5_1#$MD5_2)|($Same=Faux))
               ALERTE("Ces deux documents sont différents.")
            Fin de si
         Fin de si
      Fin de si
   Fin de si
Fin de si
```

Exemple 2

Ces exemples illustrent comment récupérer la clé digest d'un texte :

```
$key1:=Generer digest("The quick brown fox jumps over the lazy dog.";Digest MD5)
 // $key1 vaut "e4d909c290d0fb1ca068ffaddf22cbd0"
$key2:=Generer digest("The quick brown fox jumps over the lazy dog.";Digest SHA1)
 // $key2 vaut "408d94384216f890ff7a0c3528e8bed1e0b01621"
```

Exemple 3

Cet exemple n'accepte que l'utilisateur "admin" avec le mot de passe "123" ne correspondant pas à un utilisateur 4D :

```
    //Méthode base sur authentification REST
C_TEXTE($1;$2)
C_BOOLEEN($0;$3)
    //$1 : utilisateur
    //$2 : mot de passe
    //$3 : mode digest
Si($1="admin")
      Si($3)
         $0:=($2=Generer digest("123";Digest 4D))
      Sinon
         $0:=($2="123")
      Fin de si
Sinon
      $0:=Faux
Fin de si
```

Generer UUID

Generer UUID -> Résultat

Paramètre	Type		Description
Résultat	Chaîne		Nouvel UUID sous forme de texte non-canonique (32 caractères)

Description

La commande **Generer UUID** retourne un nouvel identifiant UUID de 32 caractères sous forme non-canonique.

Un UUID est un nombre d'une taille de 16 octets (128 bits). Il contient 32 caractères hexadécimaux. Il peut être exprimé sous forme non canonique (suite de 32 lettres [A-F, a-f] et/ou chiffres [0-9], par exemple 550e8400e29b41d4a716446655440000) ou sous forme canonique (groupes de 8,4,4,4,12, par exemple 550e8400-e29b-41d4-a716-446655440000).

Dans 4D, les numéros UUID peuvent être stockés dans des champs. Pour plus d'informations, reportez-vous à la section dans le manuel *Mode Développement*.

Exemple

Génération d'un UUID dans une variable :

```
C_TEXTE(MonUUID)
MonUUID:=Generer UUID
```

LANCER PROCESS EXTERNE

LANCER PROCESS EXTERNE (nomFichier {; fluxEntrée {; fluxSortie {; fluxErreur}}}{; pid})

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Chemin d'accès et arguments du fichier à lancer
fluxEntrée	Chaîne, BLOB	⇒	Flux d'entrée (stdin)
fluxSortie	Chaîne, BLOB	⇐	Flux de sortie (stdout)
fluxErreur	Chaîne, BLOB	⇐	Flux d'erreur (stderr)
pid	Entier long	⇐	Identifiant unique du process externe

Description

La commande **LANCER PROCESS EXTERNE** permet de lancer un process externe depuis 4D, sous Mac OS X et Windows.
Sous Mac OS X, cette commande donne accès à toutes les applications exécutables pouvant être lancées depuis le Terminal.

Passez dans le paramètre *nomFichier* le chemin d'accès absolu de l'application à exécuter ainsi que les arguments nécessaires, le cas échéant.
Sous Mac OS X, vous pouvez également passer uniquement le nom de l'application à exécuter, 4D utilisera alors la variable d'environnement **PATH** pour localiser l'exécutable.

Attention : Cette commande permet uniquement de lancer des applications exécutables, elle ne peut pas exécuter d'instructions dépendantes du shell (l'interpréteur de commandes). Par exemple, sous Mac OS il n'est pas possible d'utiliser cette commande pour exécuter l'instruction *echo* ou des indirections.

Le paramètre *fluxEntrée* (facultatif) contient le *stdin* du process externe. Après l'exécution de la commande, les paramètres *fluxSortie* et *fluxErreur* (s'ils sont passés) retournent respectivement le *stdout* et le *stderr* du process externe. Vous pouvez utiliser des paramètres de type BLOBs au lieu de chaînes de caractères si vous traitez des données binaires (telles que des images).

Note : Si vous utilisez la variable d'environnement *_4D_OPTION_BLOCKING_EXTERNAL_PROCESS* via la commande **FIXER VARIABLE ENVIRONNEMENT** (exécution asynchrone), les paramètres *fluxSortie* et *fluxErreur* ne sont pas retournés.

Lorsqu'il est passé, le paramètre *pid* (entier long) retourne l'identifiant unique du process (PID) affecté au niveau de l'OS, quel que soit le statut de l'option *_4D_OPTION_BLOCKING_EXTERNAL_PROCESS*. Avec cette information, il est plus facile d'interagir avec les process externes créés par la commande, par exemple pour les stopper. Si le lancement du process externe échoue, le paramètre *pid* n'est pas retourné.

Exemples sous Mac OS X

Tous les exemples suivants utilisent le Terminal de Mac OS X, accessible dans le dossier Applications/Utilitaires.

(1) Pour modifier les accès à un fichier (*chmod* est la commande Mac OS X permettant de modifier les accès des fichiers) :

```
LANCER PROCESS EXTERNE("chmod +x /dossier/monfichier.txt")
```

(2) Pour éditer un fichier texte (*cat* est la commande Mac OS X permettant d'éditer les fichiers). Dans cet exemple, le chemin d'accès absolu de la commande est passé :

```
C_TEXTE(vtentrée;vtsortie)
vtentrée:=""
LANCER PROCESS EXTERNE("/bin/cat /dossier/monfichier.txt";vtentrée;vtsortie)
```

(3) Pour récupérer la liste du contenu du dossier "Users" (*ls -l* est semblable à la commande *dir* du DOS) :

```
C_TEXTE($In;$Out)
LANCER PROCESS EXTERNE("/bin/ls -l /Users";$In;$Out)
```

(4) Pour lancer une application "graphique" indépendante, il est préférable d'utiliser la commande système *open* (dans ce cas l'instruction **LANCER PROCESS EXTERNE** a le même effet qu'un double-clic sur l'application) :

```
LANCER PROCESS EXTERNE("open /Applications/Calculator.app")
```

Exemples sous Windows

(5) Pour lancer l'application NotePad :

```
LANCER PROCESS EXTERNE("C:\\WINDOWS\\notepad.exe")
```

(6) Pour lancer l'application NotePad et ouvrir un document spécifique :

```
LANCER PROCESS EXTERNE("C:\\WINDOWS\\notepad.exe C:\\Docs\\nouveau dossier\\res.txt")
```

(7) Pour lancer l'application Microsoft® Word® et ouvrir un document spécifique (à noter l'emploi de deux ""):

```
$mondoc:="C:\\Program Files\\Microsoft Office\\Office10\\WINWORD.EXE \"C:\\Documents and
Settings\\JeanMarc\\Bureau\\MesDocs\\Nouveau dossier\\essai.xml\""
LANCER PROCESS EXTERNE($mondoc;$tIn;$tOut)
```

(8) Pour exécuter un script Perl (requiert l'installation préalable d'ActivePerl) :

```
C_TEXTE($entrée;$sortie)
```

```
FIXER VARIABLE ENVIRONNEMENT("mavariable";"valeur")
LANCER PROCESS EXTERNE("D:\\Perl\\bin\\perl.exe D:\\Perl\\eg\\cgi\\env.pl";$entrée;$sortie)
```

(9) Pour lancer une commande avec un répertoire courant défini et sans afficher la console :

```
FIXER VARIABLE ENVIRONNEMENT("_4D_OPTION_CURRENT_DIRECTORY";"C:\\4D_VCS")
FIXER VARIABLE ENVIRONNEMENT("_4D_OPTION_HIDE_CONSOLE";"true")
LANCER PROCESS EXTERNE("C:\\MesApplis\\macommande.exe")
```

(10) Pour permettre à l'utilisateur d'ouvrir un document externe sous Windows :

```
$nomdoc:=Selectionner document("";"*.*";"Choisissez le fichier à ouvrir";0)
Si(OK=1)
   FIXER VARIABLE ENVIRONNEMENT("_4D_OPTION_HIDE_CONSOLE";"true")
   LANCER PROCESS EXTERNE("cmd.exe /C start \"\" \""+$nomdoc+"\"")
Fin de si
```

(11) Les exemples suivants récupèrent la liste des process sous Windows :

```
C_ENTIER LONG($pid)
C_TEXTE($stdin;$stdout;$stderr)

LANCER PROCESS EXTERNE("tasklist";$pid) //obtenir uniquement le PID
LANCER PROCESS EXTERNE("tasklist";$stdin;$stdout;$stderr;$pid) //obtenir toutes les informations
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1. Sinon (fichier non trouvé, mémoire insuffisante, etc.), elle prend la valeur 0.

LIRE APERCU ACTIVITE

LIRE APERCU ACTIVITE (tabActivités | tabUUID ; tabDébut ; tabDurée ; tabInfo {; tabDétails}{; *})

Paramètre	Type		Description
tabActivités | tabUUID	Tableau objet, Tableau texte	⇐	Description complète des opérations (tableau objet) ou UUIDs des opérations (tableau texte)
tabDébut	Tableau texte	⇐	Heures de démarrage des opérations
tabDurée	Tableau entier long	⇐	Durées des opérations en secondes
tabInfo	Tableau texte	⇐	Description
tabDétails	Tableau objet	⇐	Détails du contexte et Sous-opérations (le cas échéant)
*	Opérateur	⇒	Si passé = Lire activité serveur

Description

La commande **LIRE APERCU ACTIVITE** retourne un ou plusieurs tableau(x) décrivant les opérations en cours de progression sur les données de 4D. Ces opérations entraînent généralement l'affichage d'une fenêtre de progression.

Cette commande permet d'obtenir un instantané des *n* opérations les plus coûteuses en temps et/ou les plus fréquentes en cours d'exécution telles que l'écriture du cache ou l'exécution de formules.

Note : Les informations retournées par la commande **LIRE APERCU ACTIVITE** sont les mêmes que celles affichées dans la page "Moniteur temps réel" (MTR) de la fenêtre d'administration de 4D Server (cf. *guide de référence de 4D Server*).

Par défaut, **LIRE APERCU ACTIVITE** traite des opérations effectuées en local (avec 4D monoposte, 4D Server ou 4D en mode distant). Avec 4D en mode distant cependant, vous pouvez également obtenir l'aperçu des opérations effectuées sur le serveur : il suffit pour cela de passer l'étoile (*) en dernier paramètre. Dans ce cas, les données du serveur seront récupérées localement. Le paramètre * est ignoré lorsque la commande est exécutée sur 4D Server ou 4D monoposte.

La commande **LIRE APERCU ACTIVITE** admet deux syntaxes :

- syntaxe utilisant uniquement un tableau objet.
- syntaxe utilisant plusieurs tableaux.

Première syntaxe : LIRE APERCU ACTIVITE (tabActivités{; *})

Avec cette syntaxe, toutes les opérations sont retournées sous forme structurée dans le tableau d'objets 4D *tabActivités*. Chaque élément du tableau est un objet construit de la manière suivante :

```
[
    {
        "message":"xxx",
        "maxValue":12321,
        "currentValue":63212,
        "interruptible:0,
        "remote":0,
        "uuid":"deadbeef",
        "taskId":xxx,
        "startTime":"2014-03-20 13:37:00:123",
        "duration":92132,
        "dbContextInfo":{
            "task_id": xxx,
            "user_name": Jean,
            "host_name": HAL,
            "task_name": "CreateIndexLocal",
            "client_uid": "DE4DB33F33F"
            "user4d_id ": 1,
            "client_version ": 123456
        },
        "dbOperationDetails":{
            table: "maTable"
            field: "Champ_1"
        },
        "subOperations":[
            {"message":"xxx",
            ...}
            ]
    },
    {...}
]
```

Voici une description de chaque propriété retournée :

- *message* (texte) : libellé de l'opération
- *maxValue* (numérique) : nombre d'itérations prévues dans l'opération (-1 si opération non itérative)
- *currentValue* (numérique) : itération courante
- *interruptible* (numérique) : opération pouvant être interrompue par l'utilisateur (0=vrai, 1=faux)
- *remote* (numérique) : opération jumelée entre le client et le serveur (0=vrai, 1=faux)
- *uuid* (texte) : identifiant UUID de l'opération
- *taskId* (numérique) : identifiant interne du process à l'origine de l'opération
- *startTime* (texte) : horaire de démarrage de l'opération au format "aaaa:mm:jj hh:mm:ss:mls"
- *duration* (numérique) : durée de l'opération en millisecondes

- *dbContextInfo* (objet) : informations relatives aux opérations traitées par le moteur de base de données. Contient les propriétés suivantes :
  - *host_name* (chaîne) : nom de l'hôte ayant lancé l'opération
  - *user_name* (chaîne) : nom de l'utilisateur 4D dont la session a lancé l'opération
  - *task_name* (chaîne) : nom du process ayant lancé l'opération
  - *task_id* (num) : numéro d'id du process ayant lancé l'opération
  - *client_uid* (chaîne) : optionnel, uuid du client ayant lancé l'opération
  - *is_remote_context* (booléen, 0 ou 1) : optionnel, indique si l'opération de base de données a été lancée par un client (valeur 1) ou par le serveur via une procédure stockée (valeur 0)
  - *user4d_id* (num) : numéro d'id de l'utilisateur 4D courant côté client
  - *client_version* (chaîne) : quatre chiffres représentant la version du moteur 4D de l'application, tels que retournés par la commande **Version application**.

  Note : client_uid et is_remote_context sont disponibles uniquement en mode client/serveur. client_uid n'est retourné que si l'opération de base de données a été démarrée sur un poste client.
- *dbOperationDetails* (objet) : propriété retournée uniquement si l'opération fait appel au moteur de base de données (c'est le cas par exemple pour les recherches et les tris). Il s'agit d'un objet contenant des informations spécifiques liées à l'opération elle-même. Les propriétés disponibles dépendent de la nature de l'opération de base de données effectuée. Ces propriétés incluent notamment :
  - table (chaîne): nom de la table impliquée dans l'opération
  - field (chaîne): nom du champ impliqué dans l'opération
  - queryPlan (chaîne): plan de recherche défini pour l'opération
  - ...
- *subOperations* (tableau) : tableau d'objets contenant les sous-opérations de l'opération courante (le cas échéant). La structure de chaque sous-élément est identique à celle de l'objet principal. Si l'opération courante n'a pas de sous-opérations, *subOperations* est vide.

Seconde syntaxe : LIRE APERCU ACTIVITE (tabUUID ;tabDébut;tabDurée; tabInfo {;tabDétails}{; *})

Avec cette syntaxe, toutes les opérations sont retournées sous forme de plusieurs tableaux synchronisés (chaque opération entraîne l'ajout d'un élément dans tous les tableaux). Les tableaux suivants sont retournés :

- *tabUUID* : contient les identifiants UUID de chaque opération (correspond à la propriété *uuid* de l'objet *tabActivités* dans la syntaxe précédente)
- *tabDébut* : contient les horaires de démarrage de chaque opération (correspond à la propriété *startTime* de l'objet *tabActivités*)
- *tabDurée* : contient les durées de chaque opération en millisecondes (correspond à la propriété *duration* de l'objet *tabActivités*)
- *tabInfo* : contient les libellés décrivant chaque opération (correspond à la propriété *message* de l'objet *tabActivités*)
- *tabDétails* (optionnel) : chaque élément de ce tableau est un objet contenant les propriétés suivantes :
  - *"dbContextInfo"* (objet) : cf. ci-dessus
  - *"dbOperationDetails"* (objet) : cf. ci-dessus
  - "subOperations". La valeur de cette propriété est un tableau objet contenant toutes les sous-opérations de l'opération courante. Si l'opération courante n'a pas de sous-opérations, la valeur de la propriété *subOperations* est un tableau vide (correspond à la propriété *subOperations* de l'objet *tabActivités*).

Exemple

Cette méthode, exécutée dans un process séparé sous 4D ou 4D Server, permet d'obtenir un instantané des opérations en progression :

```
TABLEAU TEXTE(tabUUID;0)
TABLEAU TEXTE(tabDébut;0)
TABLEAU ENTIER LONG(tabDurée;0)
TABLEAU TEXTE(tabInfo;0)

Repeter
      LIRE APERCU ACTIVITE(tabUUID;tabDébut;tabDurée;tabInfo)
      Si(Taille tableau(tabUUID)>0)
         TRACE //appel du débogueur
      Fin de si
Jusque(Faux) //Boucle infinie
```

Vous obtenez des tableaux du type :

Expression	Valeur
tabUUID	5 éléments
tabUUID	0
tabUUID{0}	""
tabUUID{1}	"99A48E47EF805F44B328D80D2E2E3197"
tabUUID{2}	"F11EE36DCED096499C11AFF8628DB0AA"
tabUUID{3}	"B5E195956EC8864A896ADD72A60C0594"
tabUUID{4}	"56DEABF7FA905F4CBFF2928F9FD89CC4"
tabUUID{5}	"CB2E848761FCC04EA3CD93C3C0CF9FD5"
tabDébut	5 éléments
tabDébut	0
tabDébut{0}	""
tabDébut{1}	"18/06/2013 - 13:59:52"
tabDébut{2}	"18/06/2013 - 13:25:21"
tabDébut{3}	"18/06/2013 - 13:59:58"
tabDébut{4}	"18/06/2013 - 13:25:21"
tabDébut{5}	"18/06/2013 - 13:59:59"
tabDurée	5 éléments
tabDurée	0
tabDurée{0}	0
tabDurée{1}	11728
tabDurée{2}	2082692
tabDurée{3}	5718
tabDurée{4}	2082707
tabDurée{5}	4750
tabInfo	5 éléments
tabInfo	0
tabInfo{0}	""
tabInfo{1}	"Chargement des données"
tabInfo{2}	"Recherche séquentielle sur Companies : 9650 sur 75662 enregistrements"
tabInfo{3}	"Tableau vers sélection : 14 sur 100"
tabInfo{4}	"Recherche séquentielle sur Companies : 9650 sur 75662 enregistrements"
tabInfo{5}	"Tableau vers sélection : 10 sur 100"

LIRE PARAMETRE MACRO

LIRE PARAMETRE MACRO (sélecteur ; paramTexte)

Paramètre	Type		Description
sélecteur	Entier long	⇨	Sélection à utiliser
paramTexte	Texte	⇦	Texte récupéré

Description

La commande **LIRE PARAMETRE MACRO** retourne dans *paramTexte* une partie ou la totalité du texte de la méthode depuis laquelle elle a été appelée.

Le paramètre *sélecteur* permet de définir le type d'information à récupérer. Vous pouvez passer l'une des constantes suivantes, placées dans le thème "**Environnement 4D**" :

Constante	Type	Valeur
Texte méthode	Entier long	1
Texte méthode surligné	Entier long	2

Si vous passez Texte méthode dans *sélecteur*, la totalité du texte de la méthode sera retourné dans *paramTexte*. Si vous passez Texte méthode surligné dans *sélecteur*, seul le texte sélectionné dans la méthode sera retourné dans *paramTexte*.

Exemple

Reportez-vous à l'exemple de **FIXER PARAMETRE MACRO**.

OUVRIR URL

OUVRIR URL (chemin {; nomApp}{; *})

Paramètre	Type		Description
chemin	Chaîne	⇒	Chemin du document ou URL à ouvrir
nomApp	Chaîne	⇒	Nom de l'application à utiliser
*	Opérateur	⇒	Si spécifié = l'URL n'est pas traduit, Si omis = l'URL est traduit

Description

La commande **OUVRIR URL** ouvre le fichier ou l'URL passé dans le paramètre *chemin* avec l'application éventuellement désignée par *nomApp*.

Le paramètre *chemin* peut contenir aussi bien un URL standard qu'un chemin d'accès de fichier. La commande accepte des ':' sous OS X, des '\' sous Windows ou un URL posix commençant par file://.

Si le paramètre *nomApp* est omis, 4D tente d'abord d'interpréter le paramètre *chemin* comme un chemin d'accès de fichier. Si c'est le cas, 4D demande au système d'ouvrir le fichier avec l'application la plus adaptée (par exemple un navigateur pour les .html, Word pour les .doc, etc.). Le paramètre *, s'il est passé, est ignoré dans ce cas.
Si le paramètre *chemin* contient un URL standard (protocoles mailto:, news:, http:, etc), 4D lance le navigateur Web par défaut et accède à l'URL. S'il n'y a pas de navigateur sur les volumes connectés à l'ordinateur, la commande ne fait rien.

Si le paramètre *nomApp* est passé, la commande interroge le système. Si une application de ce nom est installée, elle est lancée et la commande lui demande d'ouvrir l'URL ou le document spécifié.
Sous Windows, le mécanisme de reconnaissance du nom de l'application est le même que celui utilisé par la commande "Exécuter" du menu Démarrer. Vous pouvez passer par exemple :

- "iexplore" pour lancer Internet Explorer.
- "chrome" pour lancer Chrome (si installé)
- "winword" pour lancer MS Word (si installé)

Note : Vous pouvez trouver la liste des applications installées dans la *registry* à la clé suivante :
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths

Sous OS X, le mécanisme utilise la base du Finder qui indexe automatiquement toutes les applications installées. Il reconnaît toute application .app via son nom de package (avec ou sans le suffixe .app). Vous pouvez passer par exemple :

- "safari"
- "FireFox"
- "TextEdit"

Si l'application *nomApp* n'est pas trouvée, aucune erreur n'est retournée ; la commande s'exécute comme si le paramètre n'avait pas été spécifié.

4D encode automatiquement les caractères spéciaux de l'URL passé en paramètre. Si vous passez le caractère *, 4D ne traduira pas les caractères spéciaux de l'URL. Cette option permet d'accéder à ou de renvoyer des URL du type "*http://www.server.net/page.htm?q=quelquechose*".

Note : Cette commande ne fonctionne pas dans le cadre d'un process Web.

Exemple 1

Les exemples suivants illustrent les différents types de chaînes acceptées comme URLs par la commande :

```
OUVRIR URL("http://www.4d.com")
OUVRIR URL("file://C:/Users/Laurent/Documents/pending.htm")
OUVRIR URL("C:\\Users\\Laurent\\Documents\\pending.htm")
OUVRIR URL("mailto:jean_martin@4d.fr")
```

Exemple 2

Cet exemple permet de lancer l'application la plus adaptée :

```
$fichier:=Selectionner document("";"";0)
Si(OK=1)
   OUVRIR URL(Document)
Fin de si
```

Exemple 3

Vous pouvez ouvrir un même fichier texte avec différentes applications en utilisant le paramètre *nomApp* :

```
OUVRIR URL("C:\\temp\\cookies.txt") //ouvre le fichier avec Notepad
OUVRIR URL("C:\\temp\\cookies.txt";"winword") //ouvre le fichier avec MS Word (si installé)
OUVRIR URL("C:\\temp\\cookies.txt";"excel") //ouvre le fichier avec MS Excel (si installé)
```

TRAITER BALISES 4D

TRAITER BALISES 4D (templateEntrée ; résultatSortie {; param}{; param2 ; ... ; paramN})

Paramètre	Type		Description
templateEntrée	Texte, BLOB	➡	Données contenant des balises à traiter
résultatSortie	Texte, BLOB	⬅	Résultat de l'exécution du template
param	Texte, Numérique, Date, Heure, Pointeur	➡	Paramètre(s) à passer au template en exécution

Description

La commande **TRAITER BALISES 4D** provoque le traitement des balises de transformation 4D contenues dans le paramètre *templateEntrée* (champ ou variable de type Texte ou BLOB) en leur injectant optionnellement des valeurs via le(s) paramètre(s) *param* et retourne le résultat dans *résultatSortie*. Pour une description complètes de ces balises, veuillez vous reporter à la section **Balises de transformation 4D**.

Cette commande permet d'exécuter un texte de type "template" contenant des balises et des références à des expressions ou des variables 4D et de produire un résultat dépendant du contexte d'exécution et/ou des valeurs passées en paramètre.
Par exemple, vous pouvez utiliser cette commande pour générer et stocker des pages HTML à partir de **pages semi-dynamiques** contenant des balises de transformation 4D (sans qu'il soit nécessaire que le serveur Web de 4D soit démarré). Vous pouvez l'employer pour envoyer via 4D Internet Commands des courriels au format HTML contenant des traitements et/ou des références à des données contenues dans la base. Il est possible de traiter tout type de données basées sur du texte, comme le XML, le SVG ou encore le texte multistyle.

Passez les données contenant les balises à traiter dans le paramètre *templateEntrée*. Ce paramètre peut être un champ ou une variable de type Texte ou BLOB. Le type Texte sera généralement suffisant (les paramètres peuvent recevoir jusqu'à 2 Go de texte).

Note de compatibilité : A compter de la version 12 de 4D, lorsque vous utilisez des paramètres de type BLOB, la commande considère automatiquement que le jeu de caractères utilisé pour les BLOBs est MacRoman. Pour plus d'efficacité, il est fortement conseillé d'utiliser des paramètres de type Texte avec lesquels les traitements sont effectués en mode Unicode.

Toutes les balises de transformation de 4D sont prises en charge (*4DTEXT*, *4DHTML*, *4DSCRIPT*, *4DLOOP*, *4DEVAL*, etc.).

Note : En cas d'utilisation de la balise *4DINCLUDE* hors du cadre du serveur Web (process Web) :

- avec 4D en mode local et 4D Server, le dossier par défaut est le dossier contenant le fichier de structure de la base,
- avec 4D en mode distant, le dossier par défaut est le dossier contenant l'application 4D.

La commande **TRAITER BALISES 4D** prend en charge un nombre indéfini de paramètres *param* à injecter dans le code exécuté. Tout comme pour les méthodes projet, ces paramètres peuvent contenir des valeurs scalaires de type varié (texte, date, heure, entier long, réel...). Vous pouvez également utiliser des tableaux, par l'intermédiaire de pointeurs de tableaux. A l'intérieur du code traité par les balises 4D, ces paramètres sont accessibles via les arguments standard $1, $2..., comme dans les méthodes 4D (voir exemple).
Un ensemble dédié de variables locales est défini dans le contexte d'exécution de la commande **TRAITER BALISES 4D**. Ces variables peuvent être lues ou écrites pendant le traitement.

Note de compatibilité : Dans les versions précédentes de 4D, les variables locales définies dans le contexte d'appel étaient accessibles dans le contexte d'éxecution de **TRAITER BALISES 4D** en mode interprété. Ce n'est plus le cas à compter de 4D v14 R4.

Après l'exécution de la commande, le paramètre *résultatSortie* reçoit le résultat de l'exécution de *templateEntrée* incluant le traitement des balises 4D qu'il contenait, le cas échéant. Si *templateEntrée* ne contenait pas de balises 4D, le contenu de *résultatSortie* est identique à celui de *templateEntrée*.
Le paramètre *résultatSortie* peut être un champ ou une variable, il doit simplement être du même type que le paramètre *templateEntrée*.

Note : Cette commande ne provoque jamais l'appel de la **Méthode base Sur authentification Web**.

Exemple 1

Cet exemple permet de charger un document de type 'template', de traiter les balises qu'il contient puis de le stocker :

```
C_BLOB($Blob_x)
C_BLOB($blob_out)
C_TEXTE($inputText_t)
C_TEXTE($outputText_t)

DOCUMENT VERS BLOB("montemplate.txt";$Blob_x)
$inputText_t:=BLOB vers texte($Blob_x;UTF8 texte sans longueur)
TRAITER BALISES 4D($inputText_t;$outputText_t)
TEXTE VERS BLOB($outputText_t;$blob_out;UTF8 texte sans longueur)
BLOB VERS DOCUMENT($document;$blob_out)
```

Exemple 2

Cet exemple permet de générer un texte à l'aide de données dans des tableaux :

```
TABLEAU TEXTE($array;2)
$array{1}:="hello"
$array{2}:="world"
$input:="<!--#4DEVAL $1-->"
$input:=$input+"<!--#4DLOOP $2-->"
$input:=$input+"<!--#4DEVAL $2->{$2->}--> "
$input:=$input+"<!--#4DENDLOOP-->"
TRAITER BALISES 4D($input;$output;"éléments = ";->$array)
 // $output = "éléments = hello world"
```

PHP

Exécuter des scripts PHP dans 4D
PHP Executer
PHP FIXER OPTION
PHP LIRE OPTION
PHP LIRE REPONSE COMPLETE
Prise en charge des modules PHP

Exécuter des scripts PHP dans 4D

4D permet d'exécuter directement des scripts PHP, donnant ainsi accès à toute la richesse des librairies utilitaires disponibles via PHP. Ces librairies fournissent en particulier des fonctions de :

- chiffrage (MD5) et hachage,
- manipulation des fichiers ZIP,
- manipulation d'images,
- accès LDAP,
- accès COM (pilotage des documents MS Office), etc.

Cette liste n'est pas exhaustive. Pour une liste complète des modules PHP disponibles par défaut avec 4D, reportez-vous à la section **Prise en charge des modules PHP**. A noter également qu'il est possible d'installer des modules personnalisés supplémentaires.

Pour exécuter un script ou une fonction PHP, vous devez utiliser la commande **PHP Executer**. Par défaut, 4D inclut la version 5.4.11 de PHP. Les scripts exécutés doivent être compatibles avec cette version et les modules installés.

Pour une description complète des commandes et de la syntaxe PHP, veuillez vous référer à l'abondante documentation PHP disponible sur Internet. A titre d'exemple, voici des adresses de sites de référence :
http://fr.php.net/manual/fr/
http://php.developpez.com/
http://www.php.fr/
http://www.phpfrance.com/

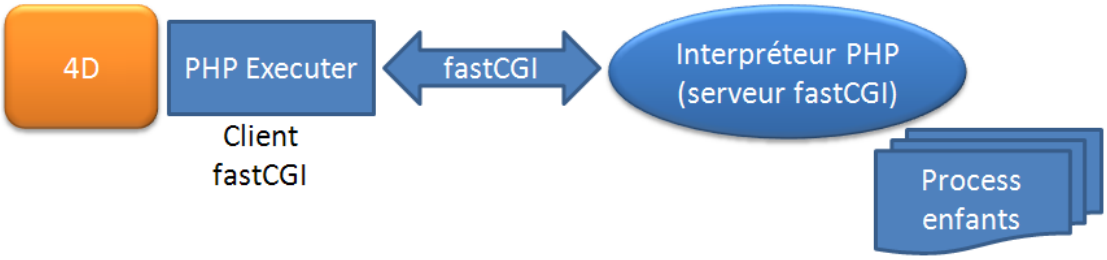
Architecture

4D fournit un interpréteur PHP compilé en FastCGI, protocole de communication de type client-serveur entre une application et un interpréteur PHP.

L'interpréteur PHP pilote un ensemble de process d'exécution système appelés "process enfants". Ces process sont dédiés au traitement des requêtes adressées par 4D. L'exécution des requêtes est synchrone. Pour des raisons d'optimisation, par défaut jusqu'à cinq process enfants peuvent être exécutés simultanément (ce nombre est modifiable via les Propriétés de la base ou la commande **FIXER PARAMETRE BASE**). Sous Mac OS, ces process sont lancés à la première requête et sont conservés en permanence par l'interpréteur PHP. Sous Windows, 4D crée les process en fonction des besoins et les recycle si nécessaire. 4D prend automatiquement en charge la gestion des process de l'interpréteur PHP fourni par défaut (lancement et fermeture).

Note : Si le programme 4D quitte inopinément alors que des process PHP enfants sont encore en activité, vous devrez les supprimer manuellement via la fenêtre de gestion des process du système.

Le schéma suivant illustre l'architecture 4D/PHP de 4D :



Cette architecture fonctionne avec un système de requêtes internes envoyées par 4D à une adresse TCP prédéfinie (adresse IP et numéro de port). Si nécessaire, par exemple si plusieurs interpréteurs PHP sont exécutés sur la même machine, cette adresse peut être modifiée via les Propriétés de la base ou la commande **FIXER PARAMETRE BASE**.

Attention : Si vous lancez deux applications 4D sur la même machine et exécutez sur chacune d'elles des instructions PHP (via la commande **PHP Executer**), vous devez impérativement modifier les ports d'écoute de l'interpréteur FastCGI PHP afin qu'ils soient différents pour chaque application. Sinon, l'exécution des instructions PHP pourra échouer de façon aléatoire, voire bloquer l'application 4D.

Utiliser un autre interpréteur PHP

Vous pouvez choisir d'utiliser un autre interpréteur PHP que celui fourni par 4D. Ce principe vous permet de conserver un même interpréteur PHP même en cas de mise à jour de 4D. En outre, il vous permet d'installer tous les modules personnalisés que vous souhaitez -- en effet, il n'est pas possible d'utiliser un fichier php.ini personnalisé avec l'interpréteur inclus de 4D. Pour utiliser des options de configurations de php autres que celles fournies par défaut, vous devez gérer un interpréteur externe.

Un interpréteur PHP personnalisé doit respecter deux conditions :

- il doit être compilé en fastCGI,
- il doit être présent sur la même machine que 4D.

Pour utiliser un interpréteur PHP personnalisé, il vous suffit de le configurer de manière à ce qu'il écoute à une adresse et un port TCP spécifiques et d'indiquer à 4D de ne pas activer l'interpréteur interne. Ces paramètres peuvent être définis soit via les Propriétés de la base, soit pour la session via la commande **FIXER PARAMETRE BASE**. Bien entendu, dans ce cas vous devez gérer vous-même le démarrage et le fonctionnement de l'interpréteur.

Le fichier d'initialisation *php.ini* personnalisé doit être placé dans le dossier **Resources** de la base. Le fichier php.ini permet notamment de déclarer l'emplacement des extensions PHP. Si ce fichier n'est pas présent lors du premier appel, 4D le crée avec les options de configuration adaptées.

Le fichier php.ini de l'interpréteur externe doit contenir les entrées suivantes :

- **auto_prepend_file** qui fournit le chemin d'accès complet au script utilitaire *4D_Execute_PHP.php*. Ce script se trouve dans [application 4D]Resources/php/Windows ou /Mac. En l'absence de cette entrée, seuls les scripts entiers peuvent être exécutés : l'appel d'une routine à l'intérieur d'un script ne fonctionnera pas.
- **display_errors** fixé à "stderr" afin que 4D puisse être informé en cas d'erreur lors de l'exécution de code PHP. Exemple :

```
; stderr - Afficher les erreurs vers STDERR (affecte uniquement les CGI/CLI)
; Pour diriger les erreurs vers STDERR pour les CGI/CLI:
display_errors = "stderr"
```

Pour plus d'informations sur la configuration des fichiers php.ini personnalisés, veuillez consulter les commentaires placés dans le fichier php.ini fourni par 4D.

PHP Executer

PHP Executer (cheminScript {; nomFonction {; résultatPHP {; param} {; param2 ; ... ; paramN}}}) -> Résultat

Paramètre	Type		Description
cheminScript	Texte	➡	Chemin d'accès au script PHP ou "" pour exécuter une fonction PHP
nomFonction	Texte	➡	Fonction PHP à exécuter
résultatPHP	Opérateur, Variable, Champ	⬅	Résultat d'exécution de la fonction PHP ou * pour ne pas recevoir de résultat
param	Texte, Booléen, Réel, Entier long, Date, Heure	➡	Paramètre(s) de la fonction PHP
Résultat	Booléen	↩	Vrai = exécution correcte, Faux = erreur d'exécution

Description

La commande **PHP Executer** permet d'exécuter un script ou une fonction PHP.

Passez dans le paramètre *cheminScript* le chemin d'accès du fichier de script PHP à exécuter. Il peut s'agir d'un chemin d'accès relatif si le fichier est situé à côté de la structure de la base ou d'un chemin absolu. Le chemin d'accès peut être exprimé en syntaxe système ou Posix.
Si vous souhaitez exécuter directement une fonction PHP standard, passez une chaîne vide ("") dans *cheminScript*. Le nom de la fonction doit être passé en deuxième paramètre.

Passez dans le paramètre *nomFonction* un nom de fonction PHP si vous souhaitez exécuter une fonction spécifique dans le script *cheminScript*. Si vous passez une chaîne vide ou omettez le paramètre *nomFonction*, le script est exécuté entièrement.

Note : PHP tient compte de la casse des caractères dans le nom de la fonction. N'utilisez pas de parenthèses, saisissez uniquement le nom de la fonction.

Le paramètre *résultatPHP* reçoit le résultat de l'exécution de la fonction PHP. Vous pouvez passer soit :

- une variable, un tableau ou un champ afin de recevoir le résultat,
- le caractère * si la fonction ne retourne pas de résultat ou si vous ne souhaitez pas le récupérer.

résultatPHP peut être de type texte, entier long, réel, booléen, date ainsi que (hormis pour les tableaux) BLOB et heure. 4D effectuera la conversion des données et les ajustements nécessaires suivant les principes décrits dans le paragraphe **Conversion des données retournées** ci-dessous.

- Si vous avez passé un nom de fonction dans *nomFonction*, *résultatPHP* recevra ce que le développeur PHP a retourné avec la commande **return** depuis le corps de la fonction.
- Si vous utilisez la commande sans passer de nom de fonction dans *nomFonction*, *résultatPHP* recevra ce que le développeur PHP a retourné avec la commande **echo** (ou une commande similaire).

Si vous appelez une fonction PHP qui attend des arguments, utilisez le(s) paramètre(s) *param1...N* pour passer une ou plusieurs valeur(s). Les valeurs doivent être séparées par des points-virgules. Vous pouvez passer des valeurs de type alpha, texte, booléen, réel, entier, entier long, date ou heure. Les images, BLOBs et objets ne sont pas admis. Vous pouvez envoyer un tableau, il est nécessaire dans ce cas de passer un pointeur sur le tableau à la commande **PHP Executer**, sinon c'est l'index courant du tableau qui est envoyé sous forme d'entier (cf. exemple). La commande accepte tous les types de tableaux sauf les tableaux pointeur, les tableaux image et les tableaux 2D.
Les paramètres *param1...N* sont envoyés au PHP au format JSON en utf-8. Ils sont automatiquement décodés avec la commande PHP **json_decode** avant d'être passés à la fonction PHP *nomFonction*.

Note : Pour des raisons techniques, la taille des paramètres passés via le protocole fast cgi ne doit pas dépasser 64 Ko. Vous devez tenir compte de cette limitation si vous utilisez des paramètres de type Texte.

La commande retourne Vrai si l'exécution s'est déroulée correctement côté 4D, c'est-à-dire si le lancement de l'environnement d'exécution, l'ouverture du script et l'établissement de la communication avec l'interpréteur PHP ont été réussis. Dans le cas contraire, une erreur est générée, que vous pouvez intercepter avec la commande **APPELER SUR ERREUR** et analyser avec **LIRE PILE DERNIERE ERREUR**.
En outre, le script lui-même peut générer des erreurs PHP. Dans ce cas, vous devez utiliser la commande **PHP LIRE REPONSE COMPLETE** afin d'analyser la source de l'erreur (voir exemple 4).

Note : PHP permet de configurer la gestion d'erreurs. Pour plus d'informations, reportez-vous par exemple à la page http://www.php.net/manual/en/errorfunc.configuration.php#ini.error-reporting.

Conversion des données retournées

Le tableau suivant précise comment 4D interprète et convertit les données retournées en fonction du type du paramètre *résultatPHP*.

Type paramètre *résultatPHP*	Traitement 4D	Exemple
BLOB	4D récupère les données reçues sans aucune modification (*).	
Texte	4D attend des données encodées en utf-8 (*). Le développeur PHP peut avoir besoin d'utiliser la commande PHP **utf8_encode**.	Exemple de script PHP : `echo utf8_encode(monTexte)`
Date	4D attend une date envoyée sous forme de chaîne au format RFC 3339 (appelé parfois DATE_ATOM en PHP). Ce format est de type "AAAA-MM-JJTHH:MM:SS", par exemple : 2005-08-15T15:52:01+00:00. 4D ignorera la partie heure et retournera la date en UTC.	
Heure	4D attend une heure envoyée sous forme de chaîne au format RFC 3339 (cf. type Date). 4D ignorera la partie date et retournera le nombre de secondes écoulées depuis minuit en considérant la date dans la zone horaire locale.	Exemple de script PHP pour envoyer 2h30'45" : `echo date( DATE_ATOM, mktime( 2,30,45))`
Entier ou Réel	4D interprète le numérique exprimé avec des chiffres, signe - ou +, exposant préfixé par 'e'. Tout caractère '.' ou ',' est interprété comme un séparateur décimal.	Exemple de script PHP : `echo -1.4e-16;`
Booléen	4D retournera Vrai s'il reçoit la chaine "true" depuis PHP ou si l'évaluation sous forme de numérique donne une valeur non nulle.	Exemple de script PHP : `echo (a==b);`
Tableau	4D considère que le tableau PHP a été retourné au format JSON.	Exemple de script PHP pour retourner un tableau de deux textes : `echo json_encode( array( "hello", "world"));`

(*) Par défaut, les en-têtes HTTP ne sont pas retournés :
- si vous utilisez **PHP Executer** en passant une fonction dans le paramètre *nomFonction*, les entêtes HTTP ne sont jamais retournés dans *résultatPHP*. Ils ne sont accessibles que via **PHP LIRE REPONSE COMPLETE**.
- si vous utilisez **PHP Executer** sans nom de fonction (*nomFonction* omis ou contenant une chaîne vide), vous pouvez retourner les en-têtes HTTP en fixant l'option PHP résultat brut à Vrai à l'aide de la commande **PHP FIXER OPTION**.

Note : Si vous devez récupérer de gros volumes de données via PHP, il est généralement plus efficace de passer par le canal du buffer *stdOut* (commande **echo** ou similaire) que par le retour de fonction. Pour plus d'informations, reportez-vous à la description de la commande **PHP LIRE REPONSE COMPLETE**.

Utiliser les variables d'environnement

Vous pouvez utiliser la commande **FIXER VARIABLE ENVIRONNEMENT** pour définir des variables d'environment utilisées par le script. Attention : après un appel à **LANCER PROCESS EXTERNE** ou **PHP Executer**, l'ensemble des variables d'environment est effacé.

Fonctions spéciales

4D propose les fonctions spéciales suivantes :

- **quit_4d_php** : permet de quitter l'interpréteur PHP et tous ses process enfants. Si un process enfant au moins est en train d'exécuter un script, l'interpréteur ne quitte pas et la commande **PHP Executer** retourne Faux.
- **relaunch_4d_php** permet de relancer l'interpréteur PHP.

A noter que l'interpréteur est relancé automatiquement à la première requête envoyée par **PHP Executer**.

Exemple 1

Appel du script "myPhpFile.php" sans fonction. Voici le contenu du script :

```
<<?php
echo 'Version php courante : ' . phpversion();
?>
```

Le code 4D suivant :

```
C_TEXTE($result)
C_BOOLEEN($isOK)
$isOK:=PHP Executer("C:\\php\\myPhpFile.php";"";$result)
ALERTE($Result)
```

... affichera "Version php courante : 5.3"

Exemple 2

Appel de la fonction *myPhpFunction* dans le script "myNewScript.php" avec des paramètres. Voici le contenu du script :

```
<?php
//...code php...
function myPhpFunction($p1, $p2) {
    return $p1 . ' '. $p2;
}
// ...code php...
?>
```

Appel avec fonction :

```
C_TEXTE($result)
C_TEXTE($param1)
C_TEXTE($param2)
C_BOOLEEN($isOk)
$param1:="Hello"
$param2 :="4D world !"
$isOk:=PHP Executer("C:\\MonDossier\\myNewScript.php";"myPhpFunction";$result;$param1;$param2 )
ALERTE($result) // Affiche "Hello 4D world!"
```

Exemple 3

Faire quitter l'interpréteur PHP :

```
$ifOk:=PHP Executer("";"quit_4d_php")
```

Exemple 4

Gestion des erreurs :

```
   // Installation de la méthode de gestion d'erreurs
phpCommError:="" // Modifiée par PHPErrorHandler
$T_saveErrorHandler :=Methode appelee sur erreur
APPELER SUR ERREUR("PHPErrorHandler")

    // Exécution du script
C_TEXTE($T_result)
Si(PHP Executer("C:\\MyScripts\\MiscInfos.php";"";$T_result))
            // Pas d'erreur, $T_Result contient le résultat
Sinon
            // Une erreur a été détectée, gérée par PHPErrorHandler
         Si(phpCommError="")
            ... // erreur PHP, utilisez PHP LIRE REPONSE COMPLETE
         Sinon
            ALERTE(phpCommError)
         Fin de si
Fin de si

 // Désinstallation de la méthode
APPELER SUR ERREUR($T_saveErrorHandler)
```

La méthode ***PHP_errHandler*** est la suivante :

```
phpCommError:=""
LIRE PILE DERNIERE ERREUR(tabCodes;tabComps;tabLibellés)
Boucle($i;1;Taille tableau(tabCodes))
      phpCommError:=phpCommError+Chaine(tabCodes{$i})+" "+tabComps{$i}+" "+tabLibellés{$i}+Caractere(Retour
chariot)
Fin de boucle
```

Exemple 5

Création dynamique par 4D d'un script avant son exécution :

```
DOCUMENT VERS BLOB("C:\\Scripts\\MonScript.php";$blobDoc)
Si(OK=1)
      $strDoc:=BLOB vers texte($blobDoc;UTF8 texte sans longueur)

      $strPosition:=Position("function2Rename";$strDoc)

      $strDoc:=Inserer chaine($strDoc;"_v2";Longueur("function2Rename")+$strPosition)

      TEXTE VERS BLOB($strDoc;$blobDoc;UTF8 texte sans longueur)
      BLOB VERS DOCUMENT("C:\\Scripts\\MonScript.php";$blobDoc)
```

```
      Si(OK#1)
         ALERTE("Erreur à la création du script")
      Fin de si
Fin de si
```

Le script est ensuite exécuté :

```
$err:=PHP Executer("C:\\Scripts\\MonScript.php";"function2Rename_v2";*)
```

Exemple 6

Récupération directe d'une valeur de type date et heure. Voici le contenu du script :

```
<?php
// ...code php...
echo date(DATE_ATOM, mktime(1, 2, 3, 4, 5, 2009));
// ...code php...
?>
```

Réception de la date côté 4D :

```
C_DATE($phpResult_date)
$result :=PHP Executer("C:\php_scripts\ReturnDate.php";"";$phpResult_date)
    //$phpResult_date vaut !05/04/2009 !

C_HEURE($phpResult_time)
$result :=PHP Executer("C:\php_scripts\ReturnDate.php";"";$phpResult_time)

    //$phpResult_time vaut ?01 :02 :03 ?
```

Exemple 7

Répartition de données dans des tableaux :

```
TABLEAU TEXTE($arText ;0)
TABLEAU ENTIER LONG($arLong ;0)
$p1 :=","
$p2 :="11,22,33,44,55"
$phpok :=PHP Executer("";"explode";$arText;$p1;$p2)
$phpok :=PHP Executer("";"explode";$arLong;$p1;$p2)

       // $arText contient les valeurs alpha "11", "22", "33", etc.
       // $arLong contient les numériques, 11, 22, 33, etc.
```

Exemple 8

Initialisation d'un tableau :

```
TABLEAU TEXTE($arText ;0)
$phpok :=PHP Executer("";"array_pad";$arText;->$arText;50;"indéfini")
       // Exécute en php : $arrTest = array_pad($arrTest, 50, 'indéfini');
       // Remplit le tableau $arText avec 50 éléments "indéfini"
```

Exemple 9

Passage de paramètres via un tableau :

```
TABLEAU ENTIER($arInt;0)
$phpok :=PHP Executer("";"json_decode";$arInt;"[13,51,69,42,7]")
       // Exécute en php : $arInt = json_decode('[13,51,69,42,7]');
       // Remplit le tableau avec des valeurs initiales
```

Exemple 10

Utilisation basique de la fonction *trim* de PHP permettant d'enlever les espaces et/ou caractères invisibles de part et d'autre d'une chaîne de caractères :

```
C_TEXTE($T_String)
$T_String:="   Bonjour  "
C_BOOLEEN($B)
$B:=PHP Executer("";"trim";$T_String;$T_String)
```

Pour plus d'informations sur la fonction *trim*, veuillez vous reporter à la documentation PHP.

PHP FIXER OPTION

PHP FIXER OPTION (option ; valeur {; *})

Paramètre	Type		Description
option	Entier long	⇒	Numéro d'option à définir
valeur	Texte, Booléen	⇒	Nouvelle valeur de l'option
*	Opérateur	⇒	Si passé : la modification ne s'applique qu'à l'appel suivant

Description

La commande **PHP FIXER OPTION** permet de définir des options spécifiques avant un appel à la commande **PHP Executer**. La portée de cette commande est le process courant.

Passez dans le paramètre *option* une constante du thème **PHP** désignant l'option à modifier et dans le paramètre *valeur*, la nouvelle valeur de l'option. Voici la description des options :

Constante	Type	Valeur	Comment
PHP privilèges	Entier long	1	Définition des privilèges utilisateur spécifiques relatifs à l'exécution du script. **Valeur(s) possible(s)** : Chaîne de la forme "Utilisateur:Mot de passe". Par exemple : "root:jd51254d"
PHP résultat brut	Entier long	2	Définition du mode de traitement des en-têtes HTTP renvoyés par PHP dans le résultat de l'exécution lorsque ce résultat est de type texte (lorsque le résultat est de type BLOB, les en-têtes sont toujours conservés). **Valeur(s) possible(s)** : Booléen. Faux (valeur par défaut) = supprimer les en-têtes HTTP du résultat. Vrai = conserver les en-têtes HTTP.

Par défaut, **PHP FIXER OPTION** définit l'option pour tous les appels à **PHP Executer** ultérieurs du process. Si vous souhaitez la définir pour le prochain appel uniquement, passez le paramètre étoile (*).

Exemple

Exécuter le script "myAdminScript.php" avec des droits d'accès Admin :

```
PHP FIXER OPTION(PHP privilèges;"admin:mypwd";*)
    `Nous passons le *, les privilèges admin seront utilisés une seule fois
C_TEXTE($result)
C_BOOLEEN($isOK)
$isOK:=PHP Executer("myAdminScript.php";$result)
Si($isOK)
   ALERTE($result)
Fin de si
```

PHP LIRE OPTION

PHP LIRE OPTION (option ; valeur)

Paramètre	Type		Description
option	Entier long	⇨	Option à lire
valeur	Texte, Booléen	⇦	Valeur courante de l'option

Description

La commande **PHP LIRE OPTION** permet de connaître la valeur courante d'une option relative à l'éxécution de scripts PHP.

Passez dans le paramètre *option* une constante du thème **PHP** désignant l'option à lire. La commande retourne dans le paramètre *valeur* la valeur courante de l'option. Vous pouvez passer une des constantes suivantes :

Constante	Type	Valeur	Comment
PHP privilèges	Entier long	1	Définition des privilèges utilisateur spécifiques relatifs à l'exécution du script. **Valeur(s) possible(s)** : Chaîne de la forme "Utilisateur:Mot de passe". Par exemple : "root:jd51254d"
PHP résultat brut	Entier long	2	Définition du mode de traitement des en-têtes HTTP renvoyés par PHP dans le résultat de l'exécution lorsque ce résultat est de type texte (lorsque le résultat est de type BLOB, les en-têtes sont toujours conservés). **Valeur(s) possible(s)** : Booléen. Faux (valeur par défaut) = supprimer les en-têtes HTTP du résultat. Vrai = conserver les en-têtes HTTP.

Note : Seul le compte utilisateur est retourné lorsque vous utilisez l'option PHP privilèges avec la commande **PHP LIRE OPTION** (le mot de passe n'est pas retourné).

Exemple

Nous souhaitons connaître le compte d'utilisateur courant :

```
C_TEXTE($userAccount)
PHP LIRE OPTION(PHP privilèges;$userAccount)
ALERTE($userAccount)
```

PHP LIRE REPONSE COMPLETE

PHP LIRE REPONSE COMPLETE (stdOut {; libellésErr ; valeursErr} {; chpsEnteteHttp {; valeursEnteteHttp}})

Paramètre	Type		Description
stdOut	Variable texte, Variable BLOB	⇦	Contenu du buffer stdOut
libellésErr	Tableau texte	⇦	Libellés des erreurs
valeursErr	Tableau texte	⇦	Valeurs des erreurs
chpsEnteteHttp	Tableau texte	⇦	Noms des en-têtes HTTP
valeursEnteteHttp	Tableau texte	⇦	Valeurs des en-têtes HTTP

Description

La commande **PHP LIRE REPONSE COMPLETE** vous permet d'obtenir des informations supplémentaires sur la réponse retournée par l'interpréteur PHP. Cette commande est particulièrement utile en cas d'erreur survenant au cours de l'exécution du script.

Le script PHP peut écrire des données dans le buffer stdOut (echo, print...). La commande retourne directement ces données dans la variable *stdOut* et applique les mêmes principes de conversion que ceux décrits dans la commande **PHP Executer**.

Les tableaux texte synchronisés *libellésErr* et *valeursErr* sont remplis lorsque l'exécution des scripts PHP provoque des erreurs. Ces tableaux fournissent des informations notamment sur l'origine, le script et la ligne de l'erreur. Ces deux tableaux sont indissociables : si *libellésErr* est passé, *valeursErr* doit être passé également.

Comme les échanges entre 4D et l'interpréteur PHP s'effectuent via FastCGI, l'interpréteur PHP fonctionne comme s'il était appelé par un serveur HTTP et envoie donc des en-têtes HTTP. Vous pouvez récupérer ces en-têtes et leurs valeurs dans les tableaux *champsEnteteHttp* et *valeursEnteteHttp*.

Prise en charge des modules PHP

Cette annexe détaille l'implémentation des modules PHP dans 4D. Les sujets suivants sont abordés :

- liste des modules standard PHP fournis par défaut avec l'interpréteur PHP de 4D
- liste des modules standard PHP non retenus par 4D.
- modifications du fichier php.ini.

Note : Si vous souhaitez installer des modules supplémentaires, vous devez utiliser un interpréteur fastcgi-php externe (cf. **Utiliser un autre interpréteur PHP**).

Modules fournis par défaut

Le tableau suivant détaille les modules PHP fournis par défaut avec 4D.

Modules génériques

Nom	Site Web	Description
BCMath	http://php.net/bc	Calculateur binaire qui prend en charge des nombres de n'importe quelle taille et précision représentés sous forme de chaînes. Exemple : `C_ENTIER LONG($valeur;$resultat)` `$valeur:=4` `$ok:=PHP Executer("";"bcpow";$resultat;$valeur;3)`
Calendar	http://php.net/calendar	Ensemble de fonctions simplifiant la conversion entre les différents formats de calendriers. Se base sur le Jour Julien. Exemple : `C_ENTIER LONG($NumberOfDays)` `$ok:=PHP Executer("";"cal_days_in_month";$NumberOfDays;1;2;2010)`
Ctype	http://php.net/ctype	Fonctions vérifiant si un caractère ou une chaîne appartient à une certaine classe de caractères, suivant la configuration locale courante Exemple : `// Vérifier que tous les caractères d'une chaîne sont des signes de ponctuation` `C_TEXTE($maChaine)` `$maChaine:=",.;/"` `$ok:=PHP Executer("";"ctype_punct";$isPunct;$maChaine)`
Date and Time	http://php.net/datetime	Récupération de la date et de l'heure depuis le serveur où le script PHP s'exécute Exemple : `//calcul de l'heure du lever du soleil à Lisbonne, Portugal` `//Latitude: 38.4 Nord` `//Longitude: 9 Ouest` `//Zénith ~= 90` `//Décalage: +1 GMT` `C_HEURE($SunriseTime)` `$ok:=PHP Executer("";"date_sunrise";$SunriseTime;0;1;38,41;-9;90;1)`
DOM (Document Object Model)	http://php.net/dom	Utilisation de documents XML via l'API DOM de PHP 5
Exif	http://php.net/exif	Travail avec les méta-données des images.
Fileinfo(*)	http://php.net/fileinfo	Détection du type de contenu et de l'encodage d'un fichier.
Filter	http://php.net/filter	Valider et filtrer les données issues de source non sécurisée, comme les entrées des utilisateurs. Exemple : `C_ENTIER LONG($filterId)` `C_TEXTE($result)` `$ok:=PHP Executer("";"filter_id";$filterId;"validate_email")` `$ok:=PHP Executer("";"filter_var";$result;"hop@123.com";$filterId)`
FTP (File Transfert Protocol)	http://php.net/ftp	Accès détaillé à un serveur FTP
Hash	http://php.net/hash	Moteur d'empreinte numérique. Permet le traitement direct ou incrémental de message de grandeur arbitraire en utilisant une variété d'algorithmes

		Exemple : `C_TEXTE($md5Result)` `$ok:=PHP Executer("";"md5";$md5Result;"ceci est ma chaîne a hacher")`
GD (Graphics Draw) Library	http://php.net/gd	Manipulation d'images
Iconv	http://php.net/iconv	Conversion de fichiers entre divers jeux de caractères
JSON (JavaScript Object Notation)	http://php.net/json	Implémentation du format d'échange de données JSON
LDAP	http://php.net/ldap	LDAP est un protocole d'accès aux "serveurs de dossiers" stockant les informations sous forme d'arborescence
LibXML	http://php.net/libxml	Librairie de fonctions et constantes XML
LibXSLT	http://php.net/xsl	Librairie de fonctions de transformation XSLT
Multibyte String	http://php.net/mbstring	Ensemble de fonctions de manipulation de chaînes qui vous permet de travailler avec les encodages multi-octets ou de traduire des jeux de caractères.
OpenSSL	http://php.net/openssl	Utilisation des fonctions de OpenSSL pour générer et vérifier les signatures, sceller (chiffrer) et ouvrir (déchiffrer) les données.
PCRE (Perl Compatible Regular Expressions)	http://php.net/pcre	Ensemble de fonctions qui implémentent les expressions rationnelles en utilisant la même syntaxe et sémantique que Perl 5 Exemple : `//Cet exemple supprime les espaces superflus d'une chaîne.` `C_TEXTE($maChaine)` `$maChaine:="foo  o  bar"` `PHP Executer("";"preg_replace";$maChaine;"/\\s\\s+/";" ";$maChaine)` `ALERTE($maChaine)` `//La chaine contient maintenant 'foo o bar' sans espaces dupliqués`
PDO (PHP Data Objects)	http://php.net/pdo	Interface d'accès à une base de données. Nécessite un driver PDO spécifique à la base de données.
PDO_SQLITE	http://php.net/pdo_sqlite	Pilote qui implémente l'interface de PHP Data Objects (PDO) pour autoriser l'accès de PHP aux bases de données SQLite 3.
Reflection	http://php.net/reflection	API de réflexion complète qui permet de faire du reverse-engineering sur les classes, les interfaces, les fonctions, les méthodes, les extensions
Phar (PHP Archive)	http://php.net/phar	Permet d'inclure une application PHP complète dans un fichier unique appelé "phar" (PHP Archive) pour faciliter son installation et sa configuration
Session	http://php.net/session	Prise en charge de sessions PHP Exemple : Les sessions sont utilisées dans les applications Web pour conserver le contexte entre chaque requête. Lorsque vous appelez **PHP Executer** dans 4D, le script PHP peut démarrer une session et stocker tout ce qui est utile à conserver comme contexte dans le tableau associé $_SESSION. Si un script PHP utilise des sessions vous devez obtenir l'ID de session retourné par PHP à l'aide de la commande **PHP LIRE REPONSE COMPLETE** et le définir avant chaque appel à **PHP Executer** à l'aide de la commande **FIXER VARIABLE ENVIRONNEMENT** `// Méthode "PHP Exécuter avec contexte"` `Si(<>PHP_Session#"")` `  FIXER VARIABLE ENVIRONNEMENT("HTTP_COOKIE";<>PHP_Session)` `Fin de si` `Si(PHP Executer($1))` `  PHP LIRE REPONSE COMPLETE($0;$errorInfos;$errorValues;$headerFields;$headerValues)` `  $idx:=Chercher dans tableau($headerFields;"Set-Cookie")` `  Si($idx>0)` `  <>PHP_Session:=$headerValues{$idx}` `  Fin de si` `Fin de si`
SimpleXML	http://php.net/simpleXML	Outils très simples et faciles à utiliser pour convertir du XML en un objet qui peut être manipulé avec ses propriétés et les itérateurs de tableaux
Sockets	http://php.net/sockets	Implémentation d'une interface bas niveau avec les fonctions de communication par socket basées sur les sockets BSD et fournit la possibilité de fonctionner aussi bien sous forme de client que de serveur.
SPL (Standard PHP Library)	http://php.net/spl	Collection d'interfaces et de classes utilitaires créés pour résoudre les problèmes usuels.
SQLite	http://php.net/sqlite	Extension pour le moteur de base de données SQLite. Ce moteur peut être embarqué.
SQLite3	http://php.net/sqlite3	Support pour les bases de données SQLite version 3
Tokenizer	http://php.net/tokenizer	Fonctions vous permettant d'écrire vos propres outils PHP d'analyse ou de modifications sans avoir à vous soucier de la spécification du langage au niveau lexical
XML (eXtensible Markup	http://php.net/xml	Analyse des documents XML

Language)		
XMLreader	http://php.net/xmlreader	Analyseur XML Pull
XMLwriter	http://php.net/xmlwriter	Génération de flux et de fichiers au format XML
Zlib	http://php.net/zlib	Lecture et écriture de fichiers compressés gzip (.gz) Exemple : `WEB LIRE ENTETE HTTP($names;$values)` `$pos:=Chercher dans tableau($names;"Accept-Encoding")` `Si($pos>0)` `  Au cas ou` `  :(Position($values{$pos};"gzip")>0)` `  WEB FIXER ENTETE HTTP("Content-Encoding: gzip")` `  PHP Executer("";"gzencode";$html;$html)` `  :(Position($values{$pos};"deflate")>0)` `  WEB FIXER ENTETE HTTP("Content-Encoding: deflate")` `  PHP Executer("";"gzdeflate";$html;$html)` `  Fin de cas` `Fin de si` `WEB ENVOYER TEXTE($html)`
Zip	http://php.net/zip	Lecture et écriture des archives compressées ZIP et des fichiers s'y trouvant

(*) Dans la version actuelle de 4D, ces modules ne sont pas disponibles sous Windows.

Modules disponibles sous Windows uniquement

Pour des raisons structurelles, les modules PHP suivants ne sont disponibles que sur la plate-forme Windows.

Nom	Site Web	Description
COM & .NET	http://php.net/com	COM (Component Object Model) est l'une des méthodes les plus utilisées pour faire communiquer des applications et des composants sur les plates-formes Windows. En outre, 4D prend en charge l'instanciation et la création d'assemblages .Net via la couche COM.
ODBC (Open DataBase Connectivity)	http://php.net/odbc	En plus du support de l'ODBC standard, l'ODBC unifié de PHP vous donne accès à diverses bases de données qui ont emprunté la sémantique des API ODBC pour implémenter leur propres API.
WDDX (Web Distributed Data eXchange)	http://php.net/wddx	Facilite les échanges de données inter-applications Web via le Web, quelle que soit la plate-forme

Modules désactivés

Les modules PHP suivants n'ont pas été implémentés dans 4D. La colonne de droite fournit la raison de cette non-implémentation :

Nom	Site Web	Cause - Solution alternative
Mimetype	http://php.net/mime-magic	Obsolète (Deprecated) - Utiliser Fileinfo
POSIX (Portable Operating System Interface)	http://php.net/posix	Obsolète (Deprecated)
Regular Expression (POSIX Extended)	http://php.net/regex	Obsolète (Deprecated) - Utiliser PCRE
Crack	http://php.net/crack	Licence restrictive
ffmpeg	http://ffmpeg-php.sourceforge.net/	Licence restrictive - Utiliser ffmpeg en ligne de commande avec LANCER PROCESS EXTERNE
Image Magick	http://php.net/manual/book.imagick.php	Licence restrictive - Utiliser GD 2
IMAP (Internet Message Access Protocol)	http://php.net/imap	Licence restrictive - Utiliser le plug-in intégré 4D Internet Commands
PDF (Portable Document Format)	http://php.net/pdf	Licence restrictive - Utiliser Haru PDF
Mysqlnd (MySQL Native Driver)	http://dev.mysql.com/downloads/connector/php-mysqlnd/	Non pertinent dans l'environnement 4D
Phar (PHP Archive)	http://php.net/phar	Non pertinent dans l'environnement 4D

Personnaliser le fichier php.ini

Le fichier "php.ini" à modifier (cf. ci-dessous) peut être situé soit dans le dossier **Resources\php** de l'application 4D (fichier par défaut) ou dans le dossier **Resources** de la base (fichier personnalisé). Reportez-vous également à la section **Exécuter des scripts PHP dans 4D**.

Attention : La modification du fichier "php.ini" doit être effectuée avec précaution et requiert une bonne connaissance de PHP. Pour plus d'informations sur la configuration des fichiers php.ini personnalisés, veuillez consulter les commentaires placés dans le fichier php.ini fourni par 4D.

Note : Si la durée du traitement PHP est relativement longue (au-delà de 30 secondes), par défaut une erreur de type 'timeout' sera retournée dans 4D et le traitement échouera. Dans ce cas, vous pouvez configurer le *timeout* par défaut afin d'allouer davantage de temps à l'exécution PHP. Pour cela, vous disposez de deux possibilités :

- fixer la variable **max_execution_time** dans le fichier "php.ini" (passez une valeur en secondes). Attention, ce paramétrage affectera tous les scripts.
- appeler la commande **set_time_limit(nbSec)** dans le script d'exécution PHP effectuant le traitement long. Passez dans *nbSec* le délai maximum alloué à l'exécution du script PHP. Ce paramétrage est conseillé car il n'affecte que le script. De manière générale, pour des raisons de sécurité il est préférable de conserver une valeur de *timeout* réduite pour les scripts PHP.

Process

- Introduction aux process
- Process 4D préemptifs
- Chercher process
- DESINSCRIRE CLIENT
- ENDORMIR PROCESS
- EXECUTER SUR CLIENT
- Executer sur serveur
- INFORMATIONS PROCESS
- INSCRIRE CLIENT
- LIRE CLIENTS INSCRITS
- Nom du process courant
- Nombre de process
- Nombre de process utilisateurs
- Nombre utilisateurs
- Nouveau process
- Numero du process courant
- Process interrompu
- REACTIVER PROCESS
- Statut du process
- SUSPENDRE PROCESS

Introduction aux process

Le multi-tâche dans 4D représente la possibilité d'exécuter simultanément plusieurs opérations de base de données distinctes. Ces opérations sont appelées des **process**.

Créer de multiples process équivaut à avoir plusieurs utilisateurs travaillant sur le même ordinateur, chacun effectuant sa tâche. Cela signifie principalement que chaque méthode peut être exécutée comme une tâche de base de données distincte.

Cette section traite des sujets suivants :

- Créer et supprimer des process
- Eléments du process
- Process utilisateurs
- Process créés par 4D
- Process locaux et globaux
- Verrouillage d'enregistrements entre process.

Note : Cette section ne traite pas des procédures stockées. Pour cela, reportez-vous à la section **Procédures stockées** dans le manuel de référence de 4D Server.

Créer et supprimer des process

Il existe plusieurs manières de créer un nouveau process :

- Exécuter une méthode en mode Développement en sélectionnant la case à cocher **Nouveau process** dans la boîte de dialogue d'exécution de méthode. La méthode choisie dans ce dialogue est la méthode process.
- Les process peuvent être démarrés par les commandes de menu. Dans l'éditeur de menus, sélectionnez la commande de menu et cochez l'option **Démarrer un process**. La méthode associée à la commande de menu est la méthode process.
- Utiliser la fonction **Nouveau process**. La méthode passée en paramètre à la fonction **Nouveau process** est la méthode process.
- Utiliser la fonction **Executer sur serveur** afin de créer une procédure stockée sur le serveur. La méthode passée en paramètre à la fonction est la méthode process.
- Utiliser la commande **APPELER WORKER**. Si le process du worker n'existe pas déjà, il est créé.

Un process peut être supprimé dans les conditions suivantes. Les deux premières sont automatiques :

- Lorsque la méthode process a terminé son exécution
- Lorsque l'utilisateur quitte la base
- Si vous stoppez le process par le langage ou utilisez le bouton **Stop** dans le débogueur
- Si vous choisissez **Tuer** dans l'Explorateur d'exécution
- Si vous appelez la commande **TUER WORKER** (pour supprimer un process worker uniquement).

Un process peut créer un autre process. Les process ne sont pas organisés hiérarchiquement. Tous les process sont égaux, et cela indépendamment du process à partir duquel ils ont été créés. Une fois qu'un process "parent" a créé un process "enfant", le process enfant pourra se poursuivre que le process parent soit toujours en cours d'exécution ou non.

Eléments d'un process

Chaque process contient certains éléments spécifiques. Il y a trois types d'éléments bien distincts dans un process :

- Les **éléments d'interface** : ce sont les éléments nécessaires à l'affichage du process.
- Les **éléments de données** : ce sont les informations liées aux données de la base.
- Les **éléments de langage** : ce sont les éléments utilisés par le langage ou importants pour le développement de l'application.

Eléments d'interface

Les éléments d'interface sont les suivants :

- **Barre de menus** : Chaque process peut avoir sa propre barre de menus courante. La barre de menus du process de premier plan est la barre de menus courante de la base.
- **Une ou plusieurs fenêtres** : Chaque process peut contrôler plusieurs fenêtres ouvertes simultanément. A l'inverse, des process peuvent n'avoir pas de fenêtre du tout.
- **Une fenêtre active (de premier plan)** : Bien qu'un process puisse disposer de plusieurs fenêtres ouvertes simultanément, chaque process n'a qu'une fenêtre active. Pour avoir plusieurs fenêtres actives à la fois, vous devez démarrer plusieurs process.

Notes :

- Par défaut, les process ne comportent pas de barre de menus, ce qui signifie que les raccourcis standard du menu **Edition** (notamment couper / copier / coller) ne sont pas disponibles dans les fenêtres du process. Lorsque vous appelez les dialogues ou les éditeurs de 4D (éditeur de formules, éditeur de recherches, **Demander**...) depuis un process, assurez-vous que l'équivalent d'un menu **Edition** est installé dans le process si vous souhaitez que l'utilisateur bénéficie des raccourcis clavier de type copier/coller.
- Les process exécutés sur le serveur (procédures stockées) et les process préemptifs ne doivent pas contenir d'éléments d'interface.

Eléments de données

Les éléments de données se réfèrent aux données de la base. Ce sont les suivants :

- **Sélection courante par table** : Chaque process a sa propre sélection courante. La même table peut avoir différentes sélections courantes dans différents process.
- **Enregistrement courant par table** : Chaque table peut avoir un enregistrement courant différent dans chaque process.

Note : Cette description des éléments de données est valide si les process sont des process globaux. Par défaut, tous les process sont globaux. Reportez-vous plus bas au paragraphe traitant de ce point.

Eléments de langage

Les éléments de langage d'un process sont tous les éléments liés à la programmation dans 4D.

- **Variables** : Chaque process a ses propres variables process. Reportez-vous à la section **Variables** pour plus d'informations. Les variables process ne sont reconnues que dans le cadre de leur process natif.
- **Table par défaut** : Chaque process a sa propre table par défaut. Cependant, notez que la commande **TABLE PAR DEFAUT** est seulement une convention de programmation.
- **Formulaires entrée et sortie** : Les formulaires entrée et sortie par défaut peuvent être choisis par programmation pour chaque table et chaque process.
- **LockedSet et ensembles process** : Chaque process a ses propres ensembles process. *LockedSet* est un ensemble process. Les ensembles process sont effacés dès que la méthode du process est terminée.
- **Appel sur erreur par process** : Chaque process a sa propre méthode de gestion d'erreurs.
- **Fenêtre de débogueur** : Chaque process peut avoir sa propre Fenêtre de débogueur.

Process utilisateur

Les process utilisateur sont des process que vous créez pour effectuer certaines tâches. Ils partagent le temps machine avec les process principaux. Par exemple, les process de connexion Web sont des process utilisateur.

L'application 4D crée également des process pour ses propres besoins. Voici les principaux process créés et gérés par 4D :

- **Process principal** : Le process principal gère les fenêtres d'affichage de l'interface utilisateur.
- **Process développement** : Le process développement gère les fenêtres et éditeurs de l'environnement de Développement. Il n'y a pas de process développement dans une base compilée ne contenant pas de code interprété.
- **Process Server Web** : Le process Server Web est créé lorsque la base est publiée sur le Web. Reportez-vous à la section **Mise en route du serveur Web et gestion des connexions** pour plus d'informations.
- **Process Gestionnaire du cache** : Le process Gestionnaire du cache gère les entrées/sorties disque de la base. Ce process est créé dès que 4D ou 4D Server est lancé.
- **Process d'indexation** : Le process d'indexation gère les index des champs de la base dans un process séparé. Ce process est créé lorsqu'un index est créé ou détruit pour un champ.
- **Process de Gestion d'événements** : Ce process est créé quand une méthode de gestion d'événement est installée par la commande **APPELER SUR EVENEMENT**. Ce process exécute la méthode d'appel sur événement installée par la commande **APPELER SUR EVENEMENT** à chaque fois qu'un événement se produit. La méthode événement est la méthode process de ce process. Elle s'exécute continuellement, même s'il n'y a pas de méthode en exécution. La gestion d'événements fonctionne aussi en mode Développement.

Process coopératifs et préemptifs

A compter de 4D v15 R5 64 bits, 4D vous permet de créer des process utilisateur préemptifs en mode compilé. Dans les versions précédentes, seuls les process utilisateur coopératifs étaient disponibles.

Lorsqu'il est exécuté en mode *préemptif*, un process est dédié à un CPU (processeur). La gestion du process est alors déléguée au système, qui peut allouer chaque CPU séparément sur une machine multi-coeurs. Lorsqu'ils sont exécutés en mode *coopératif*, tous les process sont gérés par le *thread* (process système) de l'application parente et partagent le même CPU, même sur une machine multi-coeurs.

Par conséquent, en mode préemptif, les performance globales de l'application sont améliorées, particulièrement avec des machines multi-coeurs, car de multiples threads peuvent *véritablement* être exécutés simultanément. Les gains effectifs dépendent cependant de la nature des opérations exécutées. En contrepartie, puisqu'en mode préemptif chaque thread est indépendant des autres et non géré directement par l'application, des conditions spécifiques sont à respecter dans les méthodes qui doivent être exécutées en préemptif. De plus, le mode préemptif est disponible uniquement dans certains contextes.

La gestion des process préemptifs est détaillée dans la section **Process 4D préemptifs**.

Process globaux et locaux

La portée (l'aire d'action) des process peut être globale ou locale. Par défaut, tous les process sont globaux.

Les process globaux peuvent effectuer n'importe quelle opération, y compris accéder aux données et les manipuler. Dans la plupart des cas, vous utiliserez des process globaux.

Les process locaux ne doivent être utilisés que pour des opérations qui n'accèdent pas aux données. Par exemple, vous pouvez utiliser un process local pour contrôler les éléments d'interface comme les palettes flottantes ou exécuter une méthode de gestion d'événements.

Vous spécifiez qu'un process est local via son nom. Le nom d'un process local doit commencer par le symbole dollar ($).

Attention : Si vous tentez d'accéder aux données à partir d'un process local, vous accédez aux données par l'intermédiaire du Process principal, et prenez donc le risque d'entrer en conflit avec les opérations effectuées dans ce process.

4D Server : Avec 4D Server, l'utilisation de process locaux côté client, pour des opérations qui ne nécessitent pas d'accès aux données, permet d'allouer davantage de temps machine à des tâches qui sollicitent intensivement le serveur.

Verrouillage d'enregistrements entre process

Un enregistrement est verrouillé pour un process lorsqu'un autre process l'a déjà chargé pour modification. Un enregistrement verrouillé peut être chargé par un autre process mais ne peut pas être modifié. L'enregistrement est déverrouillé seulement dans le process dans lequel l'enregistrement est modifié. Une table doit être en mode lecture/écriture pour qu'un enregistrement puisse être chargé non verrouillé. Pour plus d'informations, reportez-vous à la section **Verrouillage d'enregistrements**.

Process 4D préemptifs

4D Developer Edition 64 bit pour Windows et OS X proposent une puissante fonctionnalité : la possibilité d'exécuter du **code 4D dans un process préemptif**. Grâce à cette option, vos applications 4D compilées pourront tirer intégralement parti des machines multi-coeurs, ce qui leur permettra d'accélérer leur exécution et de prendre en charge davantage d'utilisateurs simultanément.

Qu'est-ce qu'un process préemptif ?

Lorsqu'il est exécuté en mode *préemptif*, un process est dédié à un CPU (processeur). La gestion du process est alors déléguée au système, qui peut allouer chaque CPU séparément sur une machine multi-coeurs.

Lorsqu'ils sont exécutés en mode *coopératif* (seul mode disponible dans 4D jusqu'à 4D v15 R5), tous les process sont gérés par le *thread* (process système) de l'application parente et partagent le même CPU, même sur une machine multi-coeurs.

Par conséquent, en mode préemptif, les performance globales de l'application sont améliorées, particulièrement avec des machines multi-coeurs, car de multiples threads peuvent *véritablement* être exécutés simultanément. Les gains effectifs dépendent cependant de la nature des opérations exécutées.

En contrepartie, puisqu'en mode préemptif chaque thread est indépendant des autres et non géré directement par l'application, des conditions spécifiques sont à respecter dans les méthodes qui doivent être exécutées en préemptif. De plus, le mode préemptif est disponible uniquement dans certains contextes.

Disponibilité du mode préemptif

L'utilisation du mode préemptif est disponible **uniquement dans les versions 64 bits de 4D**. Les contextes d'exécution actuellement pris en charge sont les suivants :

	Exécution en préemptif
4D Server	X
4D distant	-
4D local	X
Mode compilé	X
Mode interprété	-

Si le contexte d'exécution prend en charge le mode préemptif et si la méthode est "thread-safe", un process 4D lancé à l'aide de la commande **Nouveau process** ou **APPELER WORKER**, ou via la commande de menu "Exécuter méthode", sera exécuté dans un thread préemptif.

Sinon, si vous appelez **Nouveau process** ou **APPELER WORKER** depuis un contexte d'exécution qui n'est pas pris en charge (par exemple sur un poste 4D distant), le process sera toujours coopératif.

Note : Vous pouvez exécuter un process en mode préemptif depuis un 4D distant en démarrant une procédure stockée sur le serveur via le langage, par exemple avec **Executer sur serveur**.

Thread-safe vs thread-unsafe

Le code 4D code peut être exécuté en mode préemptif uniquement lorsque certaines conditions sont réunies. Chaque partie du code exécution (commandes, méthodes, variables, etc.) doit être compatible avec une utilisation en mode préemptif. Les éléments éligibles au mode préemptif sont qualifiés de **thread-safe** et ceux qui ne sont pas éligibles au mode préemptif sont qualifiés de **thread-unsafe**.

Note : Comme un thread est traité de manière indépendante à partir de la méthode du process parent, la totalité de la chaîne d'appel ne doit contenir aucun élément thread-unsafe ; sinon, l'exécution en mode préemptif ne sera pas possible. Ce point est abordé en détail dans le paragraphe **Quand un process est-il démarré en préemptif ?**.

La propriété "thread safe" de chaque élément dépend de l'élément lui-même :

- Commandes 4D : l'éligibilité au mode préemptif est une propriété interne. Dans le manuel *Langage* de 4D, les commandes "thread-safe" sont identifiées par l'icône  . Une grande partie des commandes 4D est d'ores et déjà éligible au mode préemptif.
- Méthodes projet : les conditions d'éligibilité au mode préemptif sont décrites dans le paragraphe **Ecrire une méthode thread-safe**.

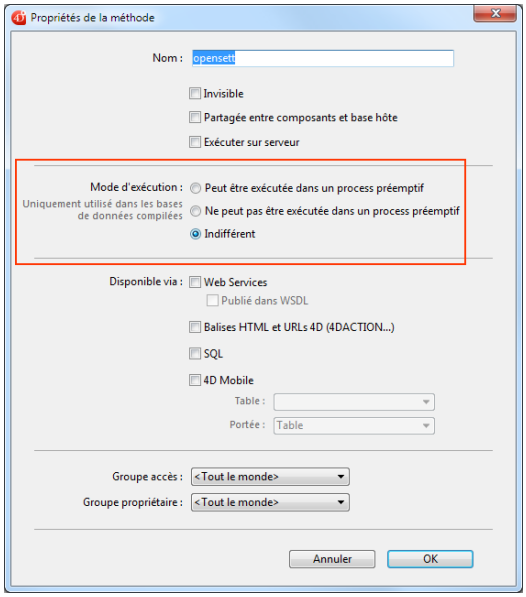
Fondamentalement, le code destiné à être exécuté dans des threads préemptifs ne peut pas appeler d'éléments ayant des interactions extérieures telles que du code de plug-in ou des variables interprocess. L'accès aux données, cependant, est possible car le serveur de données de 4D prend en charge l'exécution en mode préemptif.

Déclaration du mode d'exécution préemptif

Par défaut, 4D exécute toutes les méthodes projet de vos applications en mode coopératif. Si vous voulez bénéficier de la fonctionnalité du mode préemptif, la première étape consiste à déclarer explicitement toutes les méthodes que vous souhaitez démarrer en mode préemptif à chaque fois que c'est possible -- c'est-à-dire, les méthodes que vous estimez adaptées à une exécution dans un process préemptif. Le compilateur vérifiera que ces méthodes sont réellement thread-safe (voir pour plus d'informations). Vous pouvez également interdire le mode préemptif pour certaines méthodes, si nécessaire.

Gardez à l'esprit que déclarer une méthode "capable" d'être utilisée en préemptif la rend éligible à l'exécution dans un process préemptif mais ne garantit pas qu'elle sera effectivement exécutée en mode préemptif dans l'application. Le démarrage d'un process en mode préemptif résulte d'une évaluation effectuée par 4D en fonction de toutes les méthodes de la chaîne d'appel du process (pour plus d'informations, veuillez vous reporter au paragraphe).

Pour déclarer une méthode éligible à l'exécution en mode préemptif, vous devez sélectionner l'option de déclaration **Mode d'exécution** dans la boîte de dialogue Propriétés de la méthode :



Les options suivantes sont proposées :

- **Peut être exécutée dans un process préemptif** : En sélectionnant cette option, vous déclarez que la méthode est capable d'être exécutée dans un process préemptif et qu'elle doit donc être exécutée en mode préemptif à chaque fois que cela est possible. La propriété "preemptive" de la méthode prend pour valeur "capable".
  Lorsque cette option est sélectionnée, le compilateur de 4D vérifiera que la méthode est effectivement capable et retournera des erreurs si ce n'est pas le cas -- par exemple, si elle appelle directement ou indirectement des commandes ou d'autres méthodes qui, elles, ne peuvent pas être exécutées en mode préemptif (toute la chaîne d'appel est analysée mais les erreurs sont signalées uniquement au premier niveau). Dans ce cas, vous pourrez modifier la méthode afin de la rendre "thread-safe" ou sélectionner une autre option.
  Si l'éligibilité de la méthode au mode préemptif est confirmée par le compilateur, elle est étiquetée "thread-safe" en interne et sera exécutée en mode préemptif à chaque fois que les conditions requises seront réunies. Cette propriété atteste de l'éligibilité de la méthode au mode préemptif mais ne garantit pas que la méthode sera effectivement exécutée en mode préemptif, puisque ce mode d'exécution requiert un contexte spécifique (voir **Quand un process est-il démarré en préemptif ?**).

- **Ne peut pas être exécutée dans un process préemptif** : En sélectionnant cette option, vous déclarez que la méthode ne doit jamais être exécutée en mode préemptif, et doit par conséquent toujours être exécutée en mode coopératif. La propriété "preemptive" de la méthode prend pour valeur "incapable".
  Lorsque cette option est sélectionnée, le compilateur de 4D ne vérifiera pas la compatibilité de la méthode avec le mode préemptif ; elle sera automatiquement étiquetée "thread-unsafe" en interne (même dans le cas où elle est théoriquement compatible). Lorsqu'elle sera appelée en exécution, cette méthode "contaminera" toutes les autres méthodes dans le même thread, les forçant à s'exécuter en mode coopératif, même si elles sont elles-mêmes "thread-safe".

- **Indifférent** (défaut) : En sélectionnant cette option, vous déclarez que vous ne souhaitez pas gérer la propriété du mode préemptif pour la méthode. La propriété "preemptive" de la méthode prend pour valeur "indifferent".
  Lorsque cette option est sélectionnée, le compilateur de 4D évaluera la compatibilité de la méthode avec le mode préemptif et lui apposera l'étiquette interne "thread-safe" ou "thread-unsafe". Aucune erreur liée à l'exécution en préemptif ne sera toutefois retournée. Si la méthode est évaluée "thread-safe", à l'exécution elle n'empêchera pas l'utilisation du mode préemptif si elle est appelée dans un contexte préemptif. A l'inverse, si la méthode est évaluée "thread-unsafe", à l'exécution elle empêchera toute utilisation du mode préemptif si elle est appelée.
  A noter qu'avec cette option, quel que soit le résultat de l'évaluation de sa compatibilité avec le mode préemptif, la méthode sera toujours exécutée en mode coopératif lorsqu'elle sera appelée directement par 4D en tant que méthode parente (par exemple via la commande **Nouveau process**). La propriété "thread-safe" interne n'est prise en compte que lorsque la méthode est appelée par d'autres méthodes à l'intérieur de la chaîne d'appel.

Note : Une méthode composant déclarée "Partagée entre composants et base hôte" doit également être déclarée "capable" pour pouvoir être exécutée dans un thread préemptif par la base hôte.

Lorsque vous exportez le code de la méthode à l'aide, par exemple, de **METHODE LIRE CODE**, la propriété "preemptive" est exportée dans le commentaire "%attributes" avec la valeur "capable" ou "incapable" (la propriété n'est pas présente si l'option est "Indifférent"). Les commandes **METHODE LIRE ATTRIBUTS** et **METHODE FIXER ATTRIBUTS** permettent également de lire ou de fixer l'attribut "preemptive" avec les valeurs "indifferent", "capable" ou "incapable".

Le tableau suivant résume les effets des options de déclaration du mode préemptif :

Option	Valeur de la propriété "preemptive" (interprété)	Action du compilateur	Etiquette interne (compilé)	Mode d'exécution si la chaîne d'appel est thread-safe
Peut être exécutée dans un process préemptif	capable	Vérifie la compatibilité et retourne des erreurs si incompatible	thread-safe	Préemptif
Ne peut pas être exécutée dans un process préemptif	incapable	Aucune évaluation	thread-unsafe	Coopératif
Indifférent	indifferent	Evaluation mais pas d'erreurs	thread-safe ou thread-unsafe	Si thread-safe : préemptif ; si thread-unsafe : coopératif ; si appelée directement : coopératif

Quand un process est-il démarré en préemptif ?

Rappel : L'exécution en mode préemptif est disponible en mode compilé uniquement.

En mode compilé, lorsqu'un process est créé par la commande **Nouveau process** ou **APPELER WORKER**, 4D examine la propriété "preemptive" de la méthode du process (aussi appelée méthode *parente*) et exécute le process en mode préemptif ou coopératif en fonction de cette propriété :

- si la méthode du process est thread-safe (confirmée par le compilateur), le process est exécuté dans un thread préemptif.
- si la méthode du process est thread-unsafe, le process est exécuté dans un thread coopératif.
- si la propriété "preemptive" de la méthode du process a pour valeur "indifferent", par compatibilité le process est exécuté dans un thread coopératif (même si la méthode est en fait compatible avec le mode préemptif). A noter cependant que ce principe de compatibilité est appliqué uniquement lorsque la méthode est utilisée en tant que méthode process: une méthode déclarée "indifferent" mais étiquetée en interne "thread-safe" par le compilateur peut être appelée dans un process préemptif par une autre méthode (voir ci-dessous).

La propriété thread-safe dépend en fait de la chaîne d'appel. Si une méthode déclarée "capable" appelle une méthode thread-unsafe à n'importe lequel de ses sous-niveaux, une erreur sera retournée à la compilation : si une seule méthode de la chaîne d'appel est thread-unsafe, elle "contaminera" toutes les autres méthodes et l'exécution en préemptif sera rejetée par le compilateur. Un thread préemptif ne peut être créé que lorsque la totalité de la chaîne est thread-safe et que l'option "Peut être exécutée dans un process préemptif" a été sélectionnée pour la méthode process.
D'un autre côté, une même méthode thread-safe peut être exécutée dans un thread préemptif dans une première chaîne d'appel, et dans un thread coopératif dans une seconde chaîne d'appel.

Par exemple, considérons les méthodes projet suivantes :

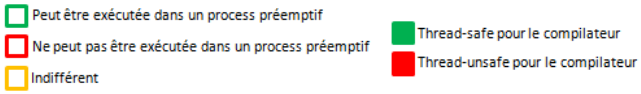
```
 //Méthode projet MyDialog
 //contient des appels d'interface : elle sera thread-unsafe en interne
$win:=Creer fenetre formulaire("tools";Form fenêtre palette)
DIALOGUE("tools")
```

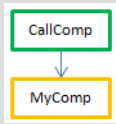
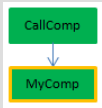
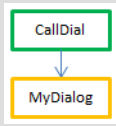
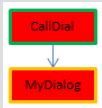
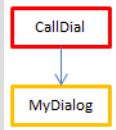
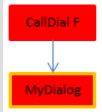
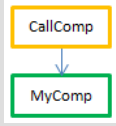
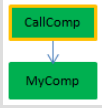
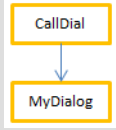
```
 //Méthode projet MyComp
 //contient des calculs simples : elle sera thread-safe en interne
C_ENTIER LONG($0;$1)
$0:=$1*2
```

```
 //Méthode projet CallDial
C_TEXTE($vName)
MyDialog
```

```
 //Méthode projet CallComp
C_ENTIER LONG($vAge)
MyCom($vAge)
```

L'exécution d'une méthode en mode préemptif dépendra de sa propriété "execution" et de la chaîne d'appel. Le tableau suivant illustre les diverses situations:



Déclaration et chaîne d'appel	Compilation	Propriété thread safety résultante	Mode d'exécution	Commentaire
	OK		Préemptif	CallComp est la méthode parente, déclarée "capable" pour le préemptif ; comme MyComp est thread-safe en interne, CallComp est thread-safe donc le process peut être préemptif
	Erreur		Exécution impossible	CallDial est la méthode parente, déclarée "capable" ; MyDialog est "indifferent". Cependant, puisque MyDialog est thread-unsafe en interne, elle contamine la chaîne d'appel. La compilation échoue à cause du conflit entre la déclaration de CallDial et ses capacités réelles. La solution est soit de modifier MyDialog afin qu'elle devienne thread-safe de manière à obtenir une exécution en préemptif, soit de changer la propriété de déclaration de CallDial pour lancer un process coopératif
	OK		Coopératif	Comme CallDial est déclarée "incapable" pour le préemptif, sa propriété interne est thread-unsafe ; son exécution sera toujours en coopératif, quel que soit le statut de MyDialog
	OK		Coopératif	Comme CallComp est la méthode parente (propriété "Indifferent"), le process est coopératif même si toute la chaîne d'appel était thread-safe
	OK		Coopératif	CallDial est la méthode parente (propriété "Indifferent"), donc le process est coopératif et la compilation réussit

Comment connaître le mode d'exécution réel

4D vous permet d'identifier le mode d'exécution des process en compilé :

- La commande **INFORMATIONS PROCESS** permet de savoir si un process est exécuté en mode préemptif ou en mode coopératif.
- L'Explorateur d'exécution et la fenêtre d'administration de 4D Server affichent des icônes spécifiques pour les process préemptifs (ainsi que pour les process worker) :

Type de process	Icône
Procédure stockée préemptif	
Process worker préemptif	
Process worker coopératif	

Ecrire une méthode thread-safe

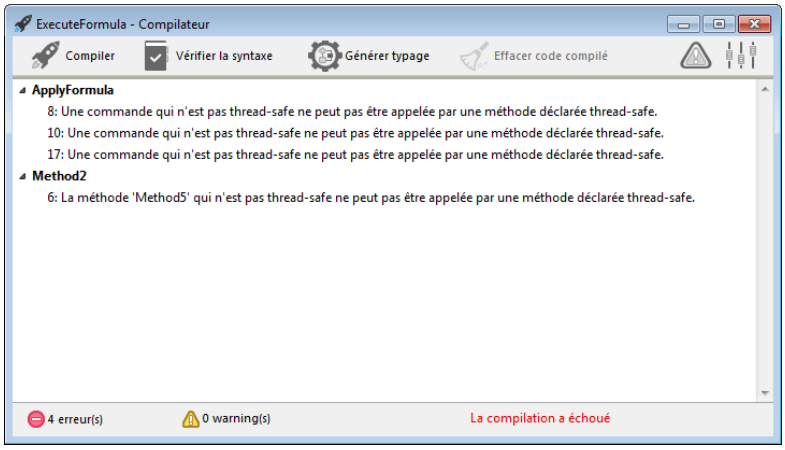
Pour être thread-safe, une méthode doit respecter les conditions suivantes :

- Sa propriété 'Mode d'exécution" doit être "Peut être exécutée dans un process préemptif" ou "Indifférent"
- Elle ne doit pas appeler de commande 4D thread-unsafe.
- Elle ne doit pas appeler de méthode projet thread-unsafe
- Elle ne doit pas appeler de plug-in
- Elle ne doit pas utiliser de blocs de code **Debut SQL/Fin SQL**
- Elle ne doit utiliser aucune variable interprocess(*)
- Elle ne doit pas appeler d'objets d'interface(**) (à quelques exceptions près, voir ci-dessous).

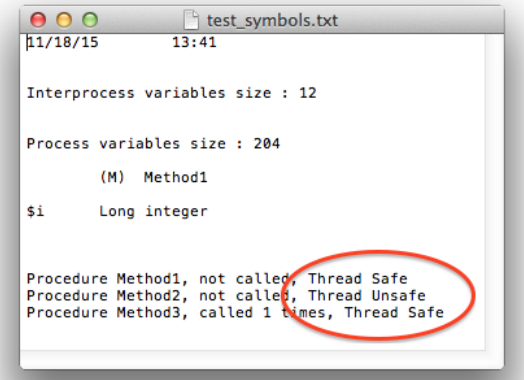
Note : Dans le cas d'une méthode "Partagée entre composants et base hôte", la propriété "Peut être exécutée dans un process préemptif" doit obligatoirement être sélectionnée.

(*) Les process de type *Worker* vous permettent d'échanger des données entre n'importe quel process, y compris des process préemptifs. Pour plus d'informations, reportez-vous à la section **A propos des workers**.
(**) La commande **APPELER FORMULAIRE** fournit une solution élégante permettant d'appeler des objets d'interface depuis un process préemptif.

Les méthodes pour lesquelles la propriété "Peut être exécutée dans un process préemptif" a été cochée seront vérifiées par 4D durant la compilation. Une erreur de compilation est générée à chaque fois que le compilateur détecte un élément non compatible avec la propriété thread-safe :



Le fichier de symboles, s'il est activé, contient également le statut thread safe pour chaque méthode :



Interface utilisateur

Puisqu'il s'agit d'accès "externes", les appels aux objets d'interface utilisateur tels que les formulaires ou le Débogueur ne ne sont pas possibles dans les threads préemptifs.

Les seuls accès à l'interface utilisateur autorisés depuis un thread préemptif sont :

- la boîte de dialogue d'erreur standard. Cette boîte de dialogue est affichée dans le process du mode utilisateur (avec 4D monoposte) ou dans le process serveur d'interface utilisateur (4D Server). Le bouton **Trace** est toutefois désactivé.
- les indicateurs de progression standard
- les boîtes de dialogue **ALERTE**, **Demander** et **CONFIRMER**. Cette boîte de dialogue est affichée dans le process du mode utilisateur (avec 4D monoposte) ou dans le process serveur d'interface utilisateur (4D Server).
  Notez que si 4D Server a été lancé en service sous Windows sans autorisation d'interaction utilisateur, les boîtes de dialogue ne sont pas affichées.

Commandes 4D thread-safe

Un nombre important de commandes 4D sont "thread-safe". Dans la documentation, l'icône  située dans la zone de propriété de commande indique que la commande est thread-safe. Vous pouvez obtenir la liste des commandes thread-safe en cliquant sur cette icône dans le manuel *Langage*.

Vous pouvez également utiliser la commande **Nom commande** qui peut retourner la valeur de la propriété "thread-safe" pour chaque commande.

Triggers

Lorsqu'une méthode utilise une commande pouvant potentiellement déclencher un trigger, le compilateur de 4D évalue la propriété "thread-safe" du trigger afin de déterminer la compatibilité de la méthode avec le mode préemptif :

```
STOCKER ENREGISTREMENT([Table_1]) //le trigger sur Table_1, s'il existe, doit être thread-safe
```

Voici la liste des commandes qui entraînent l'analyse du trigger à la compilation :

- **STOCKER ENREGISTREMENT**
- **STOCKER SUR LIEN**
- **SUPPRIMER ENREGISTREMENT**
- **SUPPRIMER SELECTION**
- **TABLEAU VERS SELECTION**
- **JSON VERS SELECTION**
- **APPLIQUER A SELECTION**
- **IMPORTER DONNEES**
- **LECTURE DIF**
- **IMPORTER ODBC**
- **LECTURE SYLK**
- **IMPORTER TEXTE**

Si la table est passée dynamiquement, il est possible que le compilateur ne soit pas en mesure de déterminer le trigger à évaluer. C'est le cas par exemple dans les situations suivantes :

```
TABLE PAR DEFAUT([Table_1])
STOCKER ENREGISTREMENT

STOCKER ENREGISTREMENT($ptrOnTable->)
STOCKER ENREGISTREMENT(Table(myMethodThatReturnsATableNumber())->)
```

Dans ce cas, tous les triggers sont évalués. Si une commande thread-unsafe est détectée dans au moins un trigger, l'ensemble est rejeté et la méthode est déclarée thread-unsafe.

Méthodes de gestion des erreurs

Les méthodes d'interception d'erreurs installées par la commande **APPELER SUR ERREUR** doivent être thread-safe si elles sont susceptibles d'être appelées depuis un process préemptif. Afin de gérer ce cas, le compilateur vérifie le caractère "thread safe" des méthodes d'appel sur erreur passées à la commande **APPELER SUR ERREUR** au moment de la compilation et retourne des erreurs si elles ne sont pas compatibles avec l'exécution en mode préemptif.

A noter que cette vérification est possible uniquement lorsque le nom de la méthode est passé en constante et n'est pas calculé, comme décrit ci-dessous :

```
APPELER SUR ERREUR("myErrMethod1") //sera vérifiée par le compilateur
APPELER SUR ERREUR("myErrMethod"+Chaine($vNum)) //ne sera pas vérifiée par le compilateur
```

De plus, à compter de 4D v15 R5, si une méthode projet d'appel sur erreur ne peut pas être appelée à l'exécution (à la suite d'un problème lié au mode préemptif ou pour toute raison, comme par exemple "méthode non trouvée"), une nouvelle erreur est générée : -10532 "Impossible d'ouvrir la méthode d'appel sur erreur 'nomMéthode'".

Compatibilité des pointeurs

Un process peut déréférencer un pointeur pour accéder à la valeur d'une autre variable process uniquement si les deux process sont coopératifs ; sinon, 4D retournera une erreur.

Exemples avec les méthodes suivantes :

Méthode 1 :

```
myVar:=42
$pid:=Nouveau process("Method2";0;"process name";->myVar)
```

Méthode 2 :

```
$value:=$1->
```

Si le process exécutant Méthode 1 ou le process exécutant Méthode 2 est préemptif, l'expression "$value:=$1->" provoquera une erreur d'exécution.

Référence de document refDoc

L'utilisation des paramètres de type *RefDoc* (référence de document ouvert, utilisée ou retournée par les commandes **Ouvrir document**, **Creer document**, **Ajouter a document**, **FERMER DOCUMENT**, **RECEVOIR PAQUET**, **ENVOYER PAQUET**) est limitée aux contextes suivants :

- Lorsqu'elle est appelée depuis un process préemptif, une référence *RefDoc* est utilisable uniquement depuis ce process préemptif.
- Lorsqu'elle est appelée depuis un process coopératif, une référence *RefDoc* est utilisable depuis n'importe quel autre process coopératif.

Pour plus d'informations sur la référence *RefDoc*, veuillez vous reporter à la section **RefDoc : numéro de référence de document**.

Chercher process

Chercher process (nom {; *}) -> Résultat

Paramètre	Type		Description
nom	Chaîne	➡	Nom du process duquel récupérer le numéro
*		➡	Retourner le numéro du process serveur
Résultat	Entier long	⮌	Numéro du process

Description

La commande **Chercher process** retourne le numéro du process dont vous passez le nom dans *nomProcess*. Si aucun process n'est trouvé, **Chercher process** retourne 0.

Le paramètre optionnel * vous permet, à partir de 4D Client, de récupérer le numéro d'un process s'exécutant sur le serveur, c'est-à-dire une procédure stockée. Dans ce cas, la valeur retournée est négative. Cette option est particulièrement utile dans le cadre de l'utilisation des commandes **LIRE VARIABLE PROCESS**, **ECRIRE VARIABLE PROCESS** et **VARIABLE VERS VARIABLE**. Pour plus d'informations, reportez-vous à la description de ces commandes.
Si la commande est exécutée avec le paramètre * à partir d'un process tournant sur le poste serveur, la valeur retournée est positive.

Exemple

Vous créez une palette flottante, fonctionnant dans un process séparé, dans lequel vous implémentez vos propres outils pour interagir avec l'environnement Développement. Par exemple, quand vous sélectionnez un élément dans une liste hérarchique de mots-clés, vous voulez coller du texte dans la fenêtre de premier plan du mode Développement. Pour cela, vous pouvez utiliser le presse-papiers, mais l'événement de collage doit se passer dans le process Développement. La petite fonction qui suit retourne le numéro du process de Développement (s'il est actif) :

```
  ` Méthode projet Numéro process Développement
  ` Numéro process Développement -> Entier long
  ` Numéro process Développement -> Numéro du process de Développement

$0:=Chercher process("Process Développement")
  ` Note: ceci peut ne pas fonctionner si le nom du process est modifié dans l'avenir
```

Avec cette fonction, la méthode projet listée ci-dessous colle le texte reçu en paramètre dans la fenêtre de premier plan du mode Développement (si c'est possible) :

```
  ` Méthode projet COLLER TEXTE EN STRUCTURE
  ` COLLER TEXTE EN STRUCTURE ( Texte)
  ` COLLER TEXTE EN STRUCTURE ( Texte à coller dans la fenêtre de Structure de premier plan )

C_TEXTE($1)
C_ENTIER LONG($vlStructurePID;$vlCompte)

$vlStructurePID:=Numero process Développement
Si($vlStructurePID #0)
  ` Mettre le texte dans le presse-papiers
   FIXER TEXTE DANS CONTENEUR($1)
  ` Générer un événement Ctrl-V / Command-V
   GENERER FRAPPE CLAVIER(Code de caractere("v");Masque touche commande;$vlStructurePID)
  ` Appeler répétitivement ENDORMIR PROCESS pour que le minuteur puisse passer
  ` l'événement au process Développement
   Boucle($vlCompte;1;5)
      ENDORMIR PROCESS(Numero du process courant;1)
   Fin de boucle
Fin de si
```

DESINSCRIRE CLIENT

DESINSCRIRE CLIENT
Ne requiert pas de paramètre

Description

La commande **DESINSCRIRE CLIENT** “désinscrit” le client 4D de 4D Server. Il doit avoir été préalablement inscrit à l’aide de la commande **INSCRIRE CLIENT**.

Si le poste client n’était pas inscrit ou si la commande est exécutée sur 4D en mode local, la commande ne fait rien.

Note : Un client 4D est automatiquement désinscrit lorsque l’application quitte.

Exemple

Reportez-vous à l’exemple de la commande **INSCRIRE CLIENT**.

Variables et ensembles système

Si le client est correctement désinscrit, la variable système OK prend la valeur 1. Si le client n’était pas inscrit, OK prend la valeur 0.

ENDORMIR PROCESS

ENDORMIR PROCESS (process ; durée)

Paramètre	Type		Description
process	Entier long	⇨	Numéro de process
durée	Réel	⇨	Durée exprimée en ticks

Description

ENDORMIR PROCESS permet d'endormir un *process* pour un certain nombre de ticks (1 tick = 1/60ème de seconde). Pendant cette période, le process endormi n'utilise pas de temps machine. Il reste cependant toujours en mémoire.

Vous pouvez endormir un process sur une durée inférieure à un tick. Par exemple, si vous passez 0,5 dans *durée*, le process sera endormi pour 1/2 tick, soit 1/120e de seconde.

Si le process est déjà endormi, cette commande l'endort à nouveau. Le paramètre *durée* n'est pas ajouté au temps restant mais le remplace. Vous pouvez passer zéro (0) dans *durée* si vous ne voulez plus endormir le process.

Si le process n'existe pas, la commande ne fait rien.

Note : Vous ne pouvez pas utiliser cette commande à partir d'un poste client pour affecter une procédure stockée sur le poste serveur (*process*<0).

Exemple 1

Reportez-vous aux exemples de la section **Verrouillage d'enregistrements**.

Exemple 2

Reportez-vous à l'exemple de la fonction **Chercher process**.

EXECUTER SUR CLIENT

EXECUTER SUR CLIENT (nomClient ; nomMéthode {; param}{; param2 ; ... ; paramN})

Paramètre	Type		Description
nomClient	Chaîne	➡	Nom d'inscription du 4D Client
nomMéthode	Chaîne	➡	Nom de la méthode à exécuter
param		➡	Paramètre(s) de la méthode

Description

La commande **EXECUTER SUR CLIENT** provoque l'exécution de la méthode *nomMéthode*, avec, éventuellement, le(s) paramètre(s) *param1... paramN*, sur le ou les 4D Client inscrit(s) sous le nom *nomClient*. Le nom d'inscription du ou des 4D Client est défini par la commande **INSCRIRE CLIENT**.

Cette commande peut être appelée depuis un 4D Client ou une procédure stockée sur 4D Server.
Si la méthode admet des paramètres, passez-les après le nom de la méthode.

L'exécution de la méthode sur le 4D Client s'effectue dans un process créé automatiquement sur le poste client, et portant le nom d'inscription du 4D Client.

Si cette commande est appelée plusieurs fois de suite pour un même 4D Client, les ordres d'exécution seront empilés. Par conséquent, les méthodes seront traitées les unes à la suite des autres : les exécutions sont asynchrones. Plus l'empilement est grand, plus la "charge de travail" est grande pour le 4D Client. Vous pouvez connaître l'état de la charge de travail de chaque client à l'aide de la commande **LIRE CLIENTS INSCRITS**.

Note : L'empilement des ordres d'exécutions ne peut être modifié ou stoppé, sauf si le 4D Client est désinscrit à l'aide de la commande **DESINSCRIRE CLIENT**.

Il est possible d'exécuter simultanément la même méthode sur plusieurs ou sur la totalité des 4D Clients inscrits : pour cela, passez le caractère joker (@) dans le paramètre *nomClient*.

Exemple 1

Vous souhaitez exécuter sur le poste client "Client1" la méthode "GénéreNums", comportant trois paramètres :

```
EXECUTER SUR CLIENT("Client1";"GénéreNums";12;$a;"Text")
```

Exemple 2

Vous souhaitez que tous les clients inscrits exécutent la méthode "VideTemp" :

```
EXECUTER SUR CLIENT("@";"VideTemp")
```

Exemple 3

Reportez-vous à l'exemple de la commande **INSCRIRE CLIENT**.

Variables et ensembles système

La variable système OK prend la valeur 1 si 4D Server a correctement reçu la requête d'exécution d'une méthode — cela ne garantit pas toutefois la bonne exécution de la méthode sur le 4D Client.

Executer sur serveur

Executer sur serveur (procédure ; pile {; nom {; param {; param2 ; ... ; paramN}}}{; *}) -> Résultat

Paramètre	Type		Description
procédure	Chaîne	➡	Procédure à exécuter dans le process
pile	Entier long	➡	Taille de la pile en octets (0 = taille par défaut)
nom	Chaîne	➡	Nom du process créé
param	Expression	➡	Paramètre(s) de la procédure
*	Opérateur	➡	Process unique
Résultat	Entier long	⬅	Numéro du process pour un process nouvellement créé ou un process déjà en cours d'exécution

Description

La commande **Executer sur serveur** lance un nouveau process sur la machine serveur (lorsqu'elle est appelée en environnement client/serveur) et retourne le numéro de ce process.

Executer sur serveur vous permet de démarrer une procédure stockée. Pour plus d'informations sur les procédures stockées, reportez-vous à la section **Procédures stockées** dans le manuel de référence de 4D Server.

Si vous appelez **Executer sur serveur** sur un poste client, la commande retourne un numéro de process négatif. Si vous appelez **Executer sur serveur** sur le poste serveur, la commande retourne un numéro de process positif. A noter que l'appel de la fonction **Nouveau process** sur le poste serveur est équivalent à l'appel de **Executer sur serveur**.

Si le process n'a pas pu être créé (par exemple s'il n'y a pas assez de mémoire), **Executer sur serveur** retourne zéro et une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

Méthode du process

Vous passez le nom de la méthode de gestion du nouveau process dans *procédure*. Une fois que 4D a défini le contexte pour le nouveau process, il démarre l'exécution de cette méthode qui devient alors la méthode du process.

Pile du process

Le paramètre *pile* permet d'indiquer la quantité de mémoire allouée pour la pile du process. Cette valeur représente la place utilisée en mémoire pour "empiler" les appels de méthode, les variables locales, les paramètres des sous-routines et les enregistrements empilés.

- Passez 0 dans *pile* pour utiliser une taille de pile par défaut, adaptée à la plupart des applications (paramétrage recommandé).
- Dans certains cas particuliers, vous pouvez souhaiter utiliser une valeur personnalisée. Elle doit être exprimée en octets. Il est conseillé de passer au minimum 64 Ko (environ 64 000 octets) et vous pouvez utiliser des valeurs au-delà de 512 Ko notamment si la chaîne d'appel dans le process (sous-routines appelant des sous-routines en cascade) est importante.

Note : La pile n'est pas la mémoire totale réservée au process. Les process se partagent la mémoire pour les enregistrements, les variables interprocess, etc. Un process utilise également de la mémoire supplémentaire pour stocker ses variables process. La pile contient diverses informations internes à 4D ; la taille de ces informations varie en fonction du nombre d'appels de méthodes imbriquées.

Note 4D Server 64 bits : La pile d'un process exécuté sur 4D Server 64 bits requiert généralement une quantité de mémoire plus importante que sur 4D Server 32 bits (environ le double). Veillez à contrôler ce paramètre lorsque votre code est destiné à être exécuté sur 4D Server 64 bits.

Nom du process :

Vous passez le nom du nouveau process dans *nom*. Avec 4D monoposte, ce nom s'affichera dans la liste des process de l'Explorateur d'exécution et sera retourné par la commande **INFORMATIONS PROCESS** appliquée à ce process. En client/serveur, ce nom apparaîtra en bleu dans la liste des **Procédures stockées** de la fenêtre principale de 4D Server.

Vous pouvez omettre ce paramètre ; dans ce cas, le nom du process sera une chaîne vide.

Attention : A la différence de la commande **Nouveau process**, vous ne devez pas avec **Executer sur serveur** créer un process local en préfixant son nom du symbole dollar (*$*). Cela fonctionnerait correctement en version monoposte, car **Executer sur serveur** se comporte comme **Nouveau process** dans cet environnement, mais, en client/serveur, cela générerait une erreur.

Paramètres de la méthode process :

Vous pouvez passer des paramètres à la méthode process. Vous pouvez le faire de la même manière que pour les sous-routines. Notez cependant qu'il y a une restriction : vous ne pouvez pas passer d'expression de type Pointeur. Rappelez-vous également que les tableaux ne peuvent pas être passés comme paramètres à une méthode. Une fois qu'elle a commencé à s'exécuter dans le contexte du nouveau process, la méthode process reçoit les valeurs des paramètres dans *$1*, *$2*, etc.

Note : Si vous passez des paramètres à la méthode process, vous devez passer le paramètre *nom*, il ne peut être omis dans ce cas.

Si vous passez un objet 4D (**C_OBJET**) comme param ou valeur de retour, la forme JSON est utilisée en utf-8 pour le serveur. Si l'objet **C_OBJET** contient des pointeurs, leur valeurs dépointées sont envoyées, pas les pointeurs eux-mêmes.

Paramètre optionnel *

Si vous passez le dernier paramètre (optionnel) *, vous indiquez à 4D de vérifier en premier lieu si un process du même nom que celui que vous avez passé dans *nom* est déjà en cours d'exécution. Si c'est le cas, 4D ne démarre pas de nouveau process et retourne le numéro du process existant.

Exemple

L'exemple suivant démontre comment l'import de données peut être accéléré de manière spectaculaire en environnement client/serveur. La méthode ***Import classique*** listée ci-dessous vous permet de mesurer combien de temps prend un import d'enregistrements basé sur la commande **IMPORTER TEXTE** :

```
   ` Méthode projet Import classique
$vhDocRef:=Ouvrir document("")
Si(OK=1)
   FERMER DOCUMENT($vhDocRef)
   FORM FIXER ENTREE([Table1];"Import")
```

```
      $vhStartTime:=Heure courante
      IMPORTER TEXTE([Table1];Document)
      $vhEndTime:=Heure courante
      ALERTE("L'opération a duré "+Chaine(0+($vhEndTime-$vhStartTime))+" secondes.")
   Fin de si
```

Avec l'import de données classique, 4D Client analyse le fichier ASCII puis, pour chaque enregistrement, crée un nouvel enregistrement, remplit les champs avec les valeurs importées et envoie l'enregistrement au poste serveur afin qu'il soit ajouté à la base. Par conséquent, de nombreuses requêtes circulent sur le réseau. Afin d'optimiser l'opération, vous pouvez utiliser des procédures stockées pour effectuer l'import localement sur le poste serveur. Le poste client charge le document dans un BLOB et déclenche une procédure stockée en passant le BLOB comme paramètre. La procédure stockée place le BLOB dans un document sur le disque du poste serveur, puis importe le document en local. L'import des données est par conséquent effectué localement à une vitesse comparable à celle d'une version monoposte de 4D, car la plupart des requêtes transitant sur le réseau ont été éliminées. Voici la méthode projet ***CLIENT IMPORT***. Lancée sur le poste client, elle déclenche l'exécution de la procédure stockée ***SERVER IMPORT***, listée à la suite :

```
   ` Méthode projet CLIENT IMPORT
   ` CLIENT IMPORT ( Pointeur ; Alpha )
   ` CLIENT IMPORT ( -> [Table] ; Formulaire entrée )

C_POINTEUR($1)
C_TEXTE($2)
C_HEURE($vhDocRef)
C_BLOB($vxData)
C_ENTIER LONG(spErrCode)

   ` Sélectionnez le document à importer
$vhDocRef:=Ouvrir document("")
Si(OK=1)
   ` Si un document était sélectionné, ne pas le garder ouvert
   FERMER DOCUMENT($vhDocRef)
   $vhStartTime:=Heure courante
   ` Essayons de le charger en mémoire
   DOCUMENT VERS BLOB(Document;$vxData)
   Si(OK=1)
   ` Si le document a pu être chargé dans le BLOB,
   ` Démarrer la procédure stockée qui va importer les données sur le poste serveur
      $spProcessID:=Executer sur serveur("SERVER IMPORT";0;"Serveur Import Services";Table($1);$2;$vxData)
   ` Nous n'avons alors plus besoin du BLOB dans ce process
      EFFACER VARIABLE($vxData)
   ` Attendons l'achèvement de l'opération effectuée par la procédure stockée
      Repeter
         ENDORMIR PROCESS(Numero du process courant;300)
         LIRE VARIABLE PROCESS($spProcessID;spErrCode;spErrCode)
         Si(Indefinie(spErrCode))
   ` Note: si la procédure stockée n'a pas initialisé sa propre instance
   ` de la variable spErrCode, il se peut qu'une variable indéfinie soit retournée
            spErrCode:=1
         Fin de si
      Jusque(spErrCode<=0)
   ` Envoyons un accusé de réception à la procédure stockée
      spErrCode:=1
      ECRIRE VARIABLE PROCESS($spProcessID;spErrCode;spErrCode)
      $vhEndTime:=Heure courante
      ALERTE("L'opération a duré "+Chaine(0+($vhEndTime-$vhStartTime))+" secondes.")
   Sinon
      ALERTE("Il n'y a pas assez de mémoire pour charger le document.")
   Fin de si
Fin de si
```

Voici la méthode projet ***SERVER IMPORT*** exécutée en tant que procédure stockée :

```
   ` Méthode projet SERVER IMPORT
   ` SERVER IMPORT ( Entier long ; Alpha ; BLOB )
   ` SERVER IMPORT ( Numéro de table ; Formulaire entrée ; Données importées )

C_ENTIER LONG($1)
C_TEXTE($2)
C_BLOB($3)
C_ENTIER LONG(spErrCode)

   ` L'opération n'est pas encore terminée, affectons 1 à spErrCode
spErrCode:=1
$vpTable:=Table($1)
FORM FIXER ENTREE($vpTable->;$2)
$vsDocName:="Fichier Import "+Chaine(1+Hasard)
SUPPRIMER DOCUMENT($vsDocName)
BLOB VERS DOCUMENT($vsDocName;$3)
IMPORTER TEXTE($vpTable->;$vsDocName)
SUPPRIMER DOCUMENT($vsDocName)
   ` L'opération est terminée, affectons 0 à spErrCode
spErrCode:=0
```

```
   ` Attendons que le poste client à l'origine de la requête ait reçu les résultats
   Repeter
      ENDORMIR PROCESS(Numero du process courant;1)
   Jusque(spErrCode>0)
```

Une fois que ces deux méthodes projet ont été implémentées dans votre base, vous pouvez effectuer un import basé sur une procédure stockée, en écrivant par exemple :

```
   CLIENT IMPORT(->[Table1];"Import")
```

Si vous réalisez quelques tests comparatifs, vous pourrez constater qu'avec ce type de méthode, l'import des enregistrements est jusqu'à 60 fois plus rapide qu'un import "classique".

INFORMATIONS PROCESS

INFORMATIONS PROCESS (process ; procNom ; procStatut ; procTemps {; procMode {; uniqueID {; origine}}})

Paramètre	Type		Description
process	Entier long	⇒	Numéro du process
procNom	Chaîne	⇐	Nom du process
procStatut	Entier long	⇐	Statut du process
procTemps	Entier long	⇐	Temps d'exécution cumulé du process en ticks
procMode	Booléen, Entier long	⇐	Si booléen : Visible (Vrai) ou Caché (Faux) Si entier long (champ de bits) : bit 0 = Visibilité, bit 1 = Exécution en préemptif
uniqueID	Entier	⇐	Numéro unique du process
origine	Entier long	⇐	Origine du process

Description

La commande **INFORMATIONS PROCESS** retourne diverses informations sur le process dont vous passez le numéro dans *process*.
Après l'appel :

- *procNom* retourne le nom du process. Quelques points sont à noter à propos du nom du process :
  - Si le process a été démarré depuis la boîte de dialogue **Exécuter une méthode** (avec l'option **Nouveau process** sélectionnée), son nom est “P_” suivi d'un numéro.
  - Si le process a été démarré à partir d'une commande de menu personnalisé dont la propriété **Démarrer un process** est sélectionnée, le nom du process est “M_” ou “ML_” suivi d'un numéro.
  - Si le process a été démarré par le serveur Web, son nom est "Web Process#" suivi d'un identifiant UUID.
  - Si un process a été stoppé (et son “espace” non encore réutilisé), son nom est encore retourné. Pour détecter si un process est stoppé, testez *procStatut*=-1 (voir ci-dessous).

- *procStatut* retourne le statut du process au moment de l'appel. Ce paramètre peut retourner l'une des valeurs fournies par les constantes prédéfinies suivantes (thème **Statut du process**) :

Constante	Type	Valeur
Détruit	Entier long	-1
Dialogue caché	Entier long	6
En attente drapeau interne	Entier long	4
En attente entrée sortie	Entier long	3
En attente événement	Entier long	2
En exécution	Entier long	0
Endormi	Entier long	1
Inexistant	Entier long	-100
Suspendu	Entier long	5

- *procTemps* retourne le cumul du temps que le process a utilisé depuis qu'il a été démarré, en ticks (1/60e de seconde).

- Le paramètre optionnel *procMode* peut être une variable de type booléen ou entier long :
  - s'il est de type booléen, il retourne Vrai si le process est visible et Faux s'il est caché.
  - s'il est de type entier long, il contient après l'exécution de la méthode un champ de bits où les deux premiers bits sont définis :
    - le bit 0 retourne la propriété de visibilité : 1 si le process est visible, et 0 s'il est caché
    - le bit 1 retourne la propriété de mode préemptif : 1 si le process est exécuté en mode préemptif, et 0 s'il est exécuté en mode coopératif.
      Note : Cette propriété est utile uniquement dans les applications 4D 64 bits, où les process peuvent être exécutés en mode préemptif ou coopératif. Pour plus d'informations, reportez-vous à la section **Process 4D préemptifs**.

- *uniqueID*, s'il est spécifié, retourne le numéro unique du process. En effet, chaque process se voit attribuer un numéro de process ainsi qu'un numéro unique de process par session. Ce dernier permet de différencier strictement deux process ou sessions de process. Il correspond au nombre de process ayant été lancés au cours de la session de l'application 4D.

- *origine*, s'il est spécifié, retourne une valeur décrivant l'origine du process. Ce paramètre peut retourner l'une des valeurs fournies par les constantes prédéfinies suivantes (thème **Type du process**) :

Constante	Type	Valeur	Comment
_o_Process Web avec contexte	Entier long	-11	
Aucun	Entier long	0	
Autre process 4D	Entier long	-10	
Autre process utilisateur	Entier long	4	
Créé par commande de menu	Entier long	2	
Créé par dialogue d'exécution	Entier long	3	
Gestionnaire Apple Event	Entier long	-7	
Gestionnaire d'événement	Entier long	-8	
Gestionnaire d'index	Entier long	-5	
Gestionnaire du cache	Entier long	-4	
Gestionnaire du port série	Entier long	-6	
Process 4D Server interne	Entier long	-18	
Process CSM	Entier long	-22	
Process d'activité	Entier long	-26	
Process de restitution	Entier long	-21	
Process de sauvegarde	Entier long	-19	
Process développement	Entier long	-2	
Process du fichier d'historique	Entier long	-20	
Process du serveur Web	Entier long	-13	
Process exécuté sur client	Entier long	-14	
Process exécuté sur serveur	Entier long	1	
Process exécution méthode SQL	Entier long	-24	
Process gestionnaire de clients	Entier long	-31	
Process interface serveur	Entier long	-15	
Process macro éditeur de méthod	Entier long	-17	
Process minuteur interne	Entier long	-25	
Process principal	Entier long	-1	
Process sur fermeture	Entier long	-16	
Process Web 4D distant	Entier long	-12	
Process Web sans contexte	Entier long	-3	
Process worker	Entier long	5	Process worker lancé par l'utilisateur
Tâche externe	Entier long	-9	

Note : Les process internes à 4D retournent une valeur négative et les process générés par l'utilisateur retournent une valeur positive.

Si le process n'existe pas, ce qui veut dire que vous n'avez pas passé un nombre inclus dans l'intervalle [1>**Nombre de process**], **INFORMATIONS PROCESS** laisse les valeurs des variables passées en paramètres inchangées.

Exemple 1

L'exemple suivant retourne le nom, le statut, et le temps écoulé dans les variables *vNom*, *vStatut*, et *vTempsPassé* pour le process courant :

```
C_ALPHA(80;vNom) ` Initialiser les variables
C_ENTIER(vStatut)
C_ENTIER(vTempsPassé)
INFORMATIONS PROCESS(Numero du process courant;vNom;vStatut;vTempsPassé)
```

Exemple 2

Voir l'exemple de la section **Méthode base Sur fermeture**.

Exemple 3

Pour connaître la visibilité et le mode d'exécution du process courant, vous pouvez écrire :

```
C_TEXTE(vNom)
C_ENTIER LONG(vStatut)
C_ENTIER LONG(vDurée)
C_ENTIER LONG(vMode)
C_BOOLEEN(estVisible)
C_BOOLEEN(estPreemptif)
INFORMATIONS PROCESS(Numero du process courant;vNom;vStatut;vDurée;vMode)
estVisible:=vMode?? 0 //vrai si visible
estPreemptif:=vMode?? 1 //vrai si préemptif
```

INSCRIRE CLIENT

INSCRIRE CLIENT (nomClient {; période}{; *})

Paramètre	Type		Description
nomClient	Chaîne	➡	Nom de la session cliente 4D
période	Entier long	➡	*** Ignoré depuis la version 11.3 ***
*	Opérateur	➡	Process local

Description

La commande **INSCRIRE CLIENT** “inscrit” un poste client 4D sous le nom *nomClient* auprès de 4D Server, afin de permettre que d’autres clients ou éventuellement 4D Server (par l’intermédiaire de procédures stockées) puissent y exécuter des méthodes à l’aide de la commande **EXECUTER SUR CLIENT**. Une fois inscrit, un client 4D peut donc exécuter une ou plusieurs méthodes pour le compte d’autres clients.

Notes :

- Vous pouvez également inscrire automatiquement chaque poste client qui se connecte à 4D Server via l'option “Inscrire les clients au démarrage pour Exécuter sur client” dans la boîte de dialogue des Préférences (cf. manuel Mode Développement).
- Lorsqu’elle est utilisée avec 4D en mode local, cette commande ne fait rien.
- Plusieurs postes clients 4D peuvent avoir le même nom d’inscription.

A l’issue de l’exécution de la commande, un process, nommé *nomClient*, est créé sur le poste client. Ce process ne peut être détruit que par la commande **DESINSCRIRE CLIENT** .
Si le paramètre optionnel * est passé, le process créé est local (4D ajoute automatiquement le signe $ au nom du process). Sinon, il est global.

Note de compatibilité : Depuis la version 11.3 de 4D, les mécanismes de communication serveur/client ont été optimisés. Désormais, le serveur envoie directement aux clients inscrits les requêtes d'exécution lorsque c'est nécessaire (technologie "push"). Le principe précédent selon lequel les clients interrogeaient périodiquement le serveur n'est plus utilisé. Le paramètre *période* est ignoré lorsqu'il est passé.

Une fois la commande exécutée, il n’est pas possible de modifier “à la volée” le nom du client 4D. Pour cela, il est nécessaire d’appeler la commande **DESINSCRIRE CLIENT** puis d’exécuter à nouveau **INSCRIRE CLIENT**.

Exemple

Les méthodes suivantes permettent de réaliser une petite messagerie entre les postes clients inscrits.

1. La méthode INSCRIPTION permet d’inscrire un client 4D et de le tenir prêt à recevoir un message de la part d’un autre client 4D :

```
  `Méthode INSCRIPTION
  `Il faut se désinscrire avant de s'inscrire sous un autre nom
DESINSCRIRE CLIENT
Repeter
   vNomPseudo:=Demander("Entrez votre nom :";"Utilisateur";"OK";"Annuler")
Jusque((OK=0)|(vNomPseudo#""))
Si(OK=0)
   ... ` Ne rien faire
Sinon
   INSCRIRE CLIENT(vNomPseudo;*)
Fin de si
```

2. L’instruction suivante permet de connaître les clients inscrits. Elle peut être placée dans la **Méthode base Sur ouverture** :

```
  ` Méthode base Sur ouverture
PrListeClient:=Nouveau process("Liste_4DClients";32000;"Liste d'inscrits")
```

3. La méthode Liste_4DClients permet de récupérer tous les clients 4D inscrits et les personnes acceptant de recevoir des messages :

```
  ` Méthode Liste_4DClients
Si(Type application=4D mode distant)
  ` Le code ci-dessous n'est valable qu'en mode client-serveur
   $Ref:=Creer fenetre(100;100;300;400;-(Fenêtre palette+Avec titre de fenêtre);"Liste d'inscrits")
   Repeter
      LIRE CLIENTS INSCRITS($ListeClient;$ListeCharge)
  `Récupération des clients inscrits dans $ListeClient
      EFFACER FENETRE($Ref)
      POSITION MESSAGE(0;0)
      Boucle($p;1;Taille tableau($ListeClient))
         MESSAGE($ListeClient{$p}+Caractere(Retour chariot))
      Fin de boucle
  `Afficher chaque seconde
      ENDORMIR PROCESS(Numero du process courant;60)
   Jusque(Faux)  ` Boucle infinie
Fin de si
```

4. La méthode Envoyer_Message permet d’envoyer un message à un autre client 4D inscrit.

```
  ` Méthode Envoyer_Message
$Destinataire:=Demander("Destinataire du message :";"")
  ` Saisir le nom d'une des personnes visibles dans la fenêtre générée par la méthode base Sur ouverture
```

```
Si(OK#0)
   $Message:=Demander("Message :") ` Contenu du message
   Si(OK#0)
      EXECUTER SUR CLIENT($Destinataire;"Afficher_Message";$Message) ` Envoi du message
   Fin de si
Fin de si
```

5. La méthode Afficher_Message affiche le message sur le poste client :

```
` Méthode Afficher_Message
C_TEXTE($1)
ALERTE($1)
```

6. Enfin, cette méthode permet à un poste client de n'être plus visible par les autres clients 4D et ne plus recevoir de message :

```
` Méthode DÉSINSCRIPTION :
DESINSCRIRE CLIENT
```

Variables et ensembles système

Si le poste client est correctement inscrit, la variable système OK prend la valeur 1. Si le poste était déjà inscrit, la commande ne fait rien et OK prend la valeur 0.

LIRE CLIENTS INSCRITS

LIRE CLIENTS INSCRITS (listeClients ; nbMéthodes)

Paramètre	Type		Description
listeClients	Tableau texte	⇐	Liste des 4D Client enregistrés
nbMéthodes	Tableau entier long	⇐	Liste des méthodes restant à exécuter

Description

La commande **LIRE CLIENTS INSCRITS** remplit deux tableaux :

- *listeClients*, qui contient la liste des clients “inscrits” à l’aide de la commande **INSCRIRE CLIENT**.
- *nbMéthodes*, qui fournit liste des “charges de travail” de chaque client. La charge de travail est le nombre de méthodes qu’un 4D Client doit encore exécuter, à la demande de la commande **EXECUTER SUR CLIENT**.

Exemple 1

Vous souhaitez obtenir la liste des clients inscrits et des méthodes restant à exécuter :

```
TABLEAU TEXTE($clients;0)
TABLEAU ENTIER LONG($nprocs;0)
LIRE CLIENTS INSCRITS($clients;$nprocs)
```

Exemple 2

Reportez-vous à l’exemple de la commande **INSCRIRE CLIENT**.

Variables et ensembles système

Si l’opération se déroule correctement, la variable système OK prend la valeur 1.

Nom du process courant

Nom du process courant -> Résultat

Paramètre	Type	Description
Résultat	Texte	Nom du process courant

Description

La commande **Nom du process courant** retourne le nom du process depuis lequel elle est appelée.

Cette commande est particulièrement utile dans le contexte des process workers (voir la section **A propos des workers**). Lorsque vous écrivez du code générique, vous pouvez l'utiliser afin d'identifier le process worker à appeler.

Exemple

Vous voulez appeler un process worker et lui passer comme paramètre le nom du process appelant :

```
APPELER WORKER(1;"monMessage";Nom du process courant;"Début :"+Chaine(vMax))
```

Nombre de process

Nombre de process -> Résultat

Paramètre	Type		Description
Résultat	Entier long	➲	Nombre total de process ouverts (y compris les process du moteur de 4D)

Description

Nombre de process retourne le nombre de process ouverts sur un poste 4D Client, 4D Server (procédures stockées) ou dans une version monoposte de 4D.

Ce nombre inclut tous les process, qu'ils soient créés par vos soins ou par 4D. Le Process principal, le Gestionnaire de cache, le Process développement, le Gestionnaire d'index ou le Process du serveur Web par exemple sont des process créés automatiquement par 4D.

La valeur retournée par **Nombre de process** comprend également les process qui ont été détruits.

Exemple

Référez-vous à l'exemple de **Statut du process** et **Méthode base Sur fermeture**.

Nombre de process utilisateurs

Nombre de process utilisateurs -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Nombre de process vivants (à l'exception de process internes)

Description

Nombre de process utilisateurs retourne le nombre courant de process "vivants" dans l'application 4D et dont le type est différent de -25 (Process minuteur interne), -31 (Process gestionnaire de clients) et -15 (Process interface serveur). Pour plus d'informations sur les types de process, reportez-vous à la commande **INFORMATIONS PROCESS** et au thème de constantes **Type du process**.

Note : Les process "vivants" sont des process dont le statut n'est ni *détruit*, ni *inexistant* (cf. commande **Statut du process**).

Nombre utilisateurs

Nombre utilisateurs -> Résultat

Paramètre	Type		Description
Résultat	Entier long	➲	Nombre d'utilisateurs connectés au serveur

Description

Lorsqu'elle est appelée depuis une procédure stockée sur le serveur, la commande **Nombre utilisateurs** retourne le nombre d'utilisateurs connectés au poste serveur.

Si le serveur exécute au moins une procédure stockée et si **Nombre utilisateurs** est appelée depuis un autre contexte (poste client, méthode Web), la commande retourne le nombre d'utilisateurs +1.

Dans le cas d'une version monoposte de 4D, **Nombre utilisateurs** retourne 1.

Nouveau process

Nouveau process (méthode ; pile {; nom {; param {; param2 ; ... ; paramN}}}{; *}) -> Résultat

Paramètre	Type		Description
méthode	Chaîne	➡	Méthode à exécuter dans le process
pile	Entier long	➡	Taille de la pile en octets (0 = taille par défaut)
nom	Chaîne	➡	Nom du process créé
param	Expression	➡	Paramètre(s) de la méthode
*	Opérateur	➡	Process unique
Résultat	Entier long	⬅	Numéro du process nouvellement créé ou du process déjà en cours d'exécution

Description

La commande **Nouveau process** lance un nouveau process (sur la même machine) et retourne le numéro de ce process.

Si le process n'a pas pu être créé, par exemple s'il n'y a pas assez de mémoire, **Nouveau process** retourne zéro et une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

Méthode du process

Vous passez le nom de la méthode de gestion du nouveau process dans *méthode*. Une fois que 4D a défini le contexte pour le nouveau process, il démarre l'exécution de cette méthode qui devient alors la méthode du process.

Pile du process

Le paramètre *pile* permet d'indiquer la quantité de mémoire allouée pour la pile du process. Cette valeur représente la place utilisée en mémoire pour "empiler" les appels de méthode, les variables locales, les paramètres des sous-routines et les enregistrements empilés.

- Passez 0 dans *pile* pour utiliser une taille de pile par défaut, adaptée à la plupart des applications (paramétrage recommandé).
- Dans certains cas particuliers, vous pouvez souhaiter utiliser une valeur personnalisée. Elle doit être exprimée en octets. Il est conseillé de passer au minimum 64 Ko (environ 64 000 octets) et vous pouvez utiliser des valeurs au-delà de 512 Ko notamment si la chaîne d'appel dans le process (sous-routines appelant des sous-routines en cascade) est importante.

Note : La pile n'est pas la mémoire totale réservée au process. Les process se partagent la mémoire pour les enregistrements, les variables interprocess, etc. Un process utilise également de la mémoire supplémentaire pour stocker ses variables process. La pile contient diverses informations internes à 4D ; la taille de ces informations varie en fonction du nombre d'appels de méthodes imbriquées.

Note 4D Server 64 bits : La pile d'un process exécuté sur 4D Server 64 bits requiert généralement une quantité de mémoire plus importante que sur 4D Server 32 bits (environ le double). Veillez à contrôler ce paramètre lorsque votre code est destiné à être exécuté sur 4D Server 64 bits.

Nom du process

Vous passez le nom du nouveau process dans *nomProcess*. Ce nom s'affichera dans la **liste des process** de l'Explorateur d'exécution et sera retourné par la commande **INFORMATIONS PROCESS**. Vous pouvez omettre ce paramètre ; dans ce cas, le nom du process sera une chaîne vide. Vous pouvez créer un process local en préfixant son nom d'un symbole dollar (*$*).

Important : Rappelez-vous que, en client/serveur, les process locaux ne doivent pas accéder aux données.

Paramètres de la méthode process

Vous pouvez passer des paramètres à la méthode process via un ou plusieurs paramètre(s) *param*. Vous pouvez le faire de la même manière que pour les sous-routines (cf. paragraphe **Passer des paramètres aux méthodes**). Une fois qu'elle a commencé à s'exécuter dans le contexte du nouveau process, la méthode process reçoit les valeurs des paramètres dans *$1*, *$2*, etc. N'oubliez pas que les tableaux ne peuvent pas être passés comme paramètres à une méthode. En outre, des considérations supplémentaires sont à prendre en compte dans le contexte de la commande **Nouveau process** :

- les pointeurs vers des tables ou des champs sont autorisés,
- les pointeurs vers des variables, en particulier des variables process et locales, sont déconseillés car les variables peuvent être indéfinies au moment où la méthode process y accède.
- si vous passez un paramètre de type objet, 4D créera dans ce cas une copie de l'objet dans le process de destination.

Note : Si vous passez des paramètres à la méthode process, vous devez passer le paramètre *nom*, il ne peut être omis dans ce cas.

Paramètre optionnel *

Si vous passez le dernier paramètre (optionnel) *, vous indiquez à 4D de vérifier en premier lieu si un process du même nom que celui que vous avez passé dans *nom* est déjà en cours d'exécution. Si c'est le cas, 4D ne démarre pas de nouveau process et retourne le numéro du process existant.

Exemple

Examinons la méthode projet suivante :

```
  ` AJOUT CLIENTS
FIXER BARRE MENUS(1)
Repeter
   AJOUTER ENREGISTREMENT([Clients];*)
Jusque(OK=0)
```

Si vous associez cette méthode projet à une commande de menu créé dans l'éditeur de barres de menus et que vous lui affectez la propriété **Démarrer un process**, 4D va automatiquement créer un nouveau process lors de l'exécution de la méthode. L'instruction **FIXER BARRE MENUS(1)** associe cette barre de menus au nouveau process. En l'absence de toute fenêtre (que vous pourriez avoir ouverte avec **Creer fenetre**), l'appel à **AJOUTER ENREGISTREMENT** en créera une automatiquement.

Si maintenant vous voulez pouvoir démarrer le process Ajout Clients lorsque vous cliquez sur un bouton situé dans un tableau de contrôle personnalisé, vous pouvez écrire :

```
   ` Méthode objet bouton bAjoutClients
$vlProcessID:=Nouveau process("Ajout Clients";0;"Ajout de clients")
```

Ce bouton fait la même chose que la commande de menu personnalisée.

Si, maintenant, lorsque la commande de menu est sélectionnée ou lorsque le bouton reçoit un clic, vous voulez que le process soit lancé s'il n'existe pas ou qu'il soit passé au premier plan s'il existe déjà, vous pouvez créer la méthode ***DEMARRER AJOUT CLIENTS*** :

```
   ` DEMARRER AJOUT CLIENTS
$vlProcessID:=Nouveau process("Ajout Clients";0;"Ajout de clients ";*)
Si($vlProcessID#0)
   PASSER AU PREMIER PLAN($vlProcessID)
Fin de si
```

La méthode objet de *bAjoutClient* devient :

```
   ` Méthode objet bouton bAjoutClients
DEMARRER AJOUT CLIENTS
```

Dans l'éditeur de barres de menus, vous remplacez ***AJOUT CLIENTS*** par la méthode ***DEMARRER AJOUT CLIENTS***. Désélectionnez l'option **Démarrer un process** pour la commande de menu.

Numero du process courant

Numero du process courant -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Numéro du process en cours d'exécution

Description

Numero du process courant retourne le numéro du process à partir duquel la fonction a été appelée.

Exemples

Référez-vous aux exemples de **ENDORMIR PROCESS** et **INFORMATIONS PROCESS**.

Process interrompu

Process interrompu -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai = le process est sur le point d'être interrompu, Faux = le process n'est pas sur le point d'être interrompu

Description

La commande **Process interrompu** retourne **Vrai** si le process dans lequel elle est appelée est sur le point d'être interrompu de manière inopinée — c'est-à-dire sans être parvenu au terme "normal" de son exécution. Cela peut se produire, par exemple, à la suite d'un appel à **QUITTER 4D**.

REACTIVER PROCESS

REACTIVER PROCESS (process)

Paramètre	Type		Description
process	Entier long	⇨	Numéro de process

Description

REACTIVER PROCESS réactive un process suspendu ou endormi. Si *process* n'est pas endormi ou suspendu, **REACTIVER PROCESS** ne fait rien.

Si *process* a été suspendu, référez-vous aux commandes **SUSPENDRE PROCESS** ou **ENDORMIR PROCESS**. Si *process* n'existe pas, cette commande ne fait rien.

Note : Vous ne pouvez pas utiliser cette commande à partir d'un poste client pour affecter une procédure stockée sur le poste serveur (*process*<0).

Statut du process

Statut du process (process) -> Résultat

Paramètre	Type		Description
process	Entier long	→	Numéro du process
Résultat	Entier long	↩	Statut du process

Description

La commande **Statut du process** retourne le statut du process dont le numéro est passé dans *process*.

Le résultat de la fonction peut être l'une des valeurs des constantes prédéfinies suivantes (thème **Statut du process**) :

Constante	Type	Valeur
Détruit	Entier long	-1
Dialogue caché	Entier long	6
En attente drapeau interne	Entier long	4
En attente entrée sortie	Entier long	3
En attente événement	Entier long	2
En exécution	Entier long	0
Endormi	Entier long	1
Inexistant	Entier long	-100
Suspendu	Entier long	5

Si le process n'existe pas (ce qui signifie le numéro que vous avez passé est hors de l'intervalle de 1 à **Nombre de process**), **Statut du process** retourne Inexistant (-100).

Exemple

L'exemple suivant retourne le nom et le numéro de référence de chaque process dans les tableaux *asProcName* et *aiProcNum*. La méthode teste si le process a été détruit. Dans ce cas, le nom et le numéro du process ne sont pas ajoutés dans le tableau :

```
$vlNbTasks:=Nombre de process
TABLEAU TEXTE(asProcName;$vlNbTasks)
TABLEAU ENTIER(aiProcNum;$vlNbTasks)
$vlActualCount:=0
Boucle($vlProcess;1;$vlNbTasks)
   Si(Statut du process($vlProcess)>=En exécution)
      $vlActualCount:=$vlActualCount+1
      INFORMATIONS PROCESS($vlProcess;asProcName{$vlActualCount};$vlState;$vlTime)
      aiProcNum{$vlActualCount}:=$vlProcess
   Fin de si
Fin de boucle
 ` Eliminer les éléments superflus
TABLEAU TEXTE(asProcName;$vlActualCount)
TABLEAU ENTIER(aiProcNum;$vlActualCount)
```

SUSPENDRE PROCESS

SUSPENDRE PROCESS (process)

Paramètre	Type		Description
process	Entier long	⇨	Numéro de process

Description

SUSPENDRE PROCESS suspend l'exécution de *process* jusqu'à ce qu'il soit remis en action par la comande **REACTIVER PROCESS**. Pendant ce temps, *process* n'utilise pas de temps machine. Lorsqu'un process est suspendu, il existe toujours en mémoire.

Si *process* est déjà suspendu, **SUSPENDRE PROCESS** ne fait rien. Si le process est endormi à l'aide de **ENDORMIR PROCESS**, le process est suspendu. S'il reçoit l'ordre **REACTIVER PROCESS**, le process redevient actif immédiatement.

Lorsqu'un process est suspendu, les fenêtres qui lui appartiennent ne sont pas saisissables. Dans ce cas, si vous ne voulez pas dérouter l'utilisateur, il faut auparavant cacher le process. Si *process* n'existe pas, cette commande ne fait rien.

Attention : Utilisez **SUSPENDRE PROCESS** seulement avec les process que vous avez créés. **SUSPENDRE PROCESS** n'a aucun effet sur le process principal.

Note : Vous ne pouvez pas utiliser cette commande à partir d'un poste client pour affecter une procédure stockée sur le poste serveur (*process*<0).

Process (Communications)

- A propos des sémaphores
- A propos des workers
- APPELER PROCESS
- APPELER WORKER
- ECRIRE VARIABLE PROCESS
- EFFACER SEMAPHORE
- LIRE VARIABLE PROCESS
- Semaphore
- Tester semaphore
- TUER WORKER
- VARIABLE VERS VARIABLE

A propos des sémaphores

Qu'est-ce qu'un sémaphore ?

Dans un programme informatique, un sémaphore est un outil permettant de protéger des actions qui ne doivent être réalisées que par un seul process ou utilisateur à la fois.

Dans 4D, le besoin classique d'usage d'un sémaphore est la modification d'un tableau interprocess : si un process est en train de modifier les valeurs du tableau, il ne faut pas qu'un autre process puisse faire la même chose en même temps. Le développeur utilise un sémaphore pour indiquer à un process qu'il ne peut réaliser la suite des opérations que si aucun autre process n'est déjà en train de réaliser les mêmes tâches. Lorsqu'un process rencontre un sémaphore, il y a trois possibilités :

- il obtient immédiatement le droit de passer
- il attend son tour, et cela jusqu'à ce qu'il obtienne le droit de passer
- il passe son chemin en abandonnant l'idée de réaliser les tâches.

Le sémaphore permet donc de protéger des parties de code. Le sémaphore ne laisse passer qu'un process à la fois et interdit l'accès tant que le process détenteur du droit d'usage ne redonne pas son droit en libérant le sémaphore.

Commandes de manipulation des sémaphores

Dans 4D, vous posez un sémaphore en appelant la fonction **Semaphore**. Pour libérer un sémaphore, vous appelez la commande **EFFACER SEMAPHORE**.

La fonction **Semaphore** a un comportement très particulier car elle réalise potentiellement deux actions en même temps :

- si le sémaphore est déjà attribué, la fonction retourne **Vrai**
- si le sémaphore n'est pas attribué, la fonction attribue le sémaphore au process et répond **Faux** dans le même temps.

Ce principe de double action effectuée par la même commande permet de garantir qu'aucune opération extérieure ne peut s'insérer entre le test du sémaphore et son attribution.

La commande **Tester semaphore** permet de savoir si un sémaphore est déjà attribué ou non. Cette commande est principalement utile dans le cadre d'opérations longues, comme par exemple la clôture annuelle de la comptabilité, où **Tester semaphore** permet de contrôler l'interface pour interdire l'accès à certaines opérations telles que l'ajout de données comptables.

Comment utiliser les sémaphores ?

Il est recommandé d'utiliser les sémaphores en respectant les principes suivants :

- un sémaphore doit être posé et libéré dans la même méthode,
- l'exécution du code protégé par le sémaphore doit être la plus courte possible,
- le code doit être temporisé à l'aide du paramètre *nbTicks* de la fonction **Semaphore** pour attendre la libération du sémaphore.

Voici le code type d'utilisation d'un sémaphore :

```
Tant que(Semaphore("MonSemaphore";300))
   APPELER 4D
Fin tant que
 // placer ici le code protégé par le sémaphore
EFFACER SEMAPHORE("MonSemaphore")
```

Un sémaphore non libéré peut provoquer le blocage d'une partie de la base. Poser et libérer le sémaphore au sein de la même méthode permet pratiquement d'éliminer ce risque.

Réduire au maximum le code protégé par le sémaphore augmente la fluidité de l'application et évite que le sémaphore soit un goulot d'étranglement.

Enfin, l'utilisation du paramètre optionnel *nbTicks* de la commande **Semaphore** est indispensable pour optimiser l'attente de la libération d'un sémaphore. Avec ce paramètre, la commande fonctionne de la manière suivante :

- le process attendra au maximum ce nombre de ticks (300 dans l'exemple) que le sémaphore soit disponible, sans que l'exécution du code passe à la ligne suivante,
- si le sémaphore est libéré avant la fin du délai imparti, il est immédiatement attribué au process (**Semaphore** retourne Faux) et l'exécution du code reprend
- si le sémaphore n'est pas libéré avant la fin du délai imparti, l'exécution du code reprend.

La commande réalise également une priorisation des demandes en mettant en place une file d'attente. Ainsi, le premier process demandant un sémaphore sera le premier à l'obtenir.

A noter que la durée d'attente est à régler en fonction des spécificités de l'application.

Sémaphore local ou global

Il y a deux types de sémaphores dans 4D : les sémaphores locaux et les sémaphores globaux.

- Un sémaphore local est visible par tous les process d'un même poste et seulement sur ce poste. Vous déclarez un sémaphore local en préfixant son nom avec le signe dollar ($). Les sémaphores locaux permettent de contrôler des opérations entre les différents process exécutés sur le même poste. Par exemple, un sémaphore local peut être utilisé pour gérer les accès à un tableau interprocess appelé par tous les process d'une base de données mono-utilisateur ou d'un poste client.

- Un sémaphore global est visible par tous les utilisateurs et tous les process. Les sémaphores globaux permettent de contrôler des opérations entre les postes clients d'une base multi-utilisateurs.

Le principe de fonctionnement des sémaphores globaux et locaux est identique. Leur différence réside uniquement dans leur portée, c'est-à-dire leur visibilité. En client-serveur, les sémaphores globaux sont visibles pour tous les process de tous les postes clients et du serveur. Un sémaphore local n'est visible que pour les process du poste sur lequel il a été créé.

Avec 4D, les sémaphores globaux et locaux ont la même portée car il n'y a qu'un seul utilisateur. Cependant, si votre base est utilisée dans les deux environnements, n'hésitez pas à employer des sémaphores globaux et locaux, en fonction de vos besoins.

Note : Les sémaphores locaux sont recommandés lorsque l'usage d'un sémaphore est nécessaire pour gérer un aspect local à un client de l'application, comme par exemple l'interface ou un tableau de valeurs interprocess. L'utilisation d'un sémaphore global provoquerait dans ce cas non seulement des échanges réseau inutiles, mais en plus pourrait affecter inutilement d'autres postes clients. Le sémaphore local évitera ces effets indésirables.

A propos des workers

Présentation

Les **process workers** constituent un moyen simple et puissant d'échanger des informations entre les process. Cette fonctionnalité est basée sur un système de messagerie asynchrone, permettant d'appeler spécifiquement des process ou des formulaires et de leur demander d'exécuter des méthodes avec des paramètres dans leur propre contexte.

Note : Une méthode projet peut également être exécutée avec des paramètres dans le contexte de tout formulaire à l'aide de la commande **APPELER FORMULAIRE**.

Un "worker" peut être "engagé" par tout process (via la commande **APPELER WORKER**) afin d'exécuter des méthodes projet avec des paramètres dans le contexte du worker, ce qui permet d'accéder à des informations partagées.

Cette fonctionnalité répond aux besoins suivants en matière de communication interprocess dans 4D :

- Comme les workers peuvent être des process coopératifs ou préemptifs, ils représentent une solution idéale pour la communication interprocess entre les process préemptifs (les variables interprocess ne sont pas utilisables avec les process préemptifs).
- Ils fournissent également une alternative simple à l'emploi de sémaphores, qui peuvent s'avérer difficiles à mettre en oeuvre et à utiliser (cf. **A propos des sémaphores**).

Note : Bien qu'elles aient été conçues principalement pour les besoins liés à la communication interprocess dans le contexte des process préemptifs (accessibles en version 64 bits uniquement), les commandes **APPELER WORKER** et **APPELER FORMULAIRE** sont disponibles dans les versions 32 bits et peuvent être utilisées avec des process en mode coopératif.

Utilisation des workers

Un **worker** est utilisé pour demander à un process d'exécuter des méthodes projet. Un worker est constitué des éléments suivants :

- un nom unique, également utilisé pour nommer son process associé
- un process associé, qui peut exister ou non à un instant donné
- une boîte aux lettres
- une méthode de démarrage (optionnel)

Vous demandez à un worker d'exécuter une méthode projet en appelant la commande **APPELER WORKER**. Le worker et sa boîte aux lettres sont créés à la première utilisation ; son process associé est également lancé automatiquement à la première utilisation. Si le process du worker est tué par la suite, la boîte aux lettres restera toutefois ouverte et tout nouveau message posté entraînera le lancement d'un nouveau process worker.

L'animation suivante illustre cette séquence :

A la différence d'un process créé par la commande **Nouveau process**, un process worker reste vivant après la fin de l'exécution de sa méthode process. Cela signifie que toutes les méthodes exécutées par le même worker le seront dans le même process, qui maintient son contexte (variables process, enregistrement courant, sélection courante...). Par conséquent, les méthodes exécutées successivement accèderont aux mêmes informations et donc les partageront, permettant ainsi aux process de communiquer entre eux. La boîte aux lettres du worker traite les appels successifs de façon asynchrone.

APPELER WORKER encapsule le nom de la méthode ainsi que ses paramètres dans un message qui est envoyé dans la boîte aux lettres du worker. Le process du worker est alors démarré s'il n'existe pas déjà, et traite le contenu du message (il exécute la méthode avec ses éventuels paramètres). Cela signifie que **APPELER WORKER** va généralement rendre la main avant que la méthode ait terminé son exécution (les traitements sont asynchrones). Pour cette raison, **APPELER WORKER** ne retourne pas de valeur. Si vous avez besoin qu'un worker retourne des valeurs au process qui l'a appelé (*callback*), vous devez utiliser à nouveau **APPELER WORKER** afin de repasser les informations attendues au process appelant. Bien entendu dans ce cas, l'appelant lui-même doit être un worker.

Il n'est pas possible d'utiliser **APPELER WORKER** pour exécuter une méthode dans un process créé par la commande **Nouveau process**. Seul un process worker a une boîte aux lettres et donc, peut être appelé par **APPELER WORKER**. Notez qu'un process créé par **Nouveau process** peut appeler un worker, mais ne peut pas être appelé en retour.

Les process workers peuvent être créés sur 4D Server via des procédures stockées : par exemple, vous pouvez utiliser la commande **Executer sur serveur** pour exécuter une méthode qui appelle la commande **APPELER WORKER**.

Un process worker est détruit par l'appel de la commande **TUER WORKER**, qui vide la boîte aux lettres du worker et demande au process associé de ne plus traiter de message, puis de terminer son exécution après la fin du traitement de sa tâche en cours.

La méthode de démarrage est la méthode utilisée pour créer initialement le worker (première utilisation). Si **APPELER WORKER** est appelée avec un paramètre *méthode* vide, la méthode de démarrage du worker est automatiquement réutilisée comme méthode à exécuter.

Le process principal, créé par 4D à l'ouverture de la base pour l'interface utilisateur et le mode application est un process worker et peut être appelé par **APPELER WORKER**. A noter que le nom du process principal peut varier en fonction de la langue de 4D, mais a toujours le numéro 1 ; par conséquent il est plus pratique de le désigner par son numéro de process au lieu de son nom lorsque vous utilisez **APPELER WORKER**.

Identification des process workers

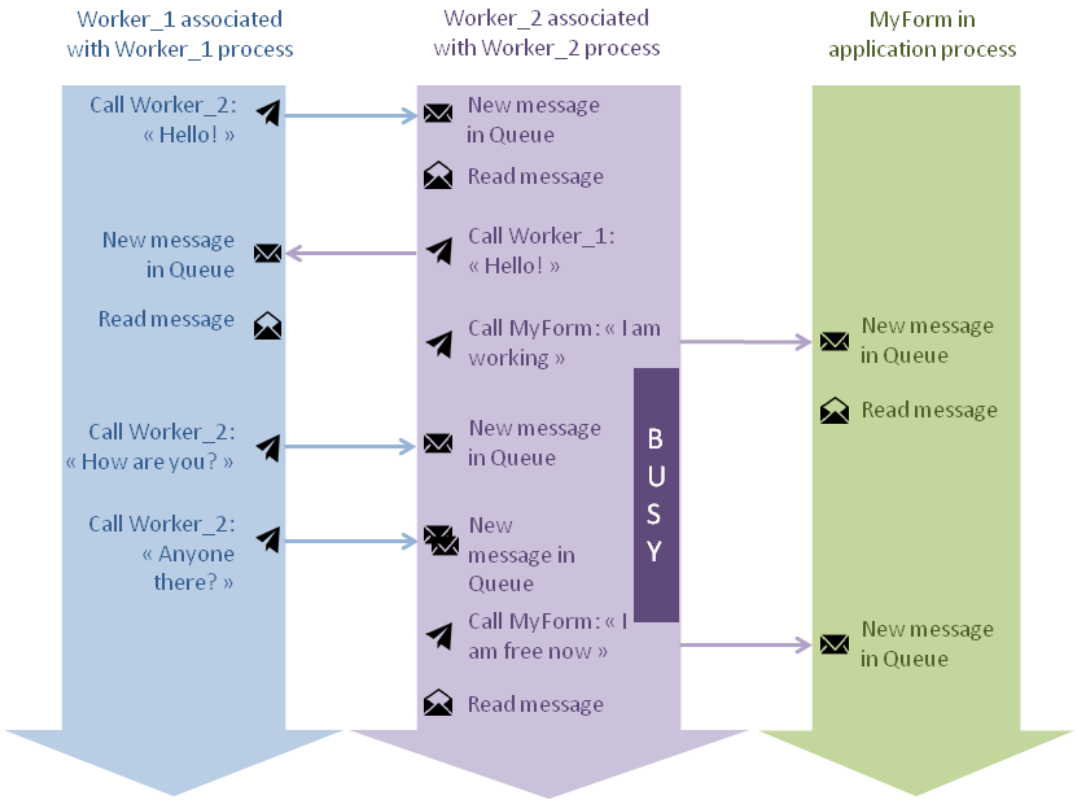
Tous les process workers, à l'exception du process principal, ont le type Process worker (5) retourné par la commande **INFORMATIONS PROCESS**.

Des icônes spécifiques identifient les process workers dans l'Explorateur d'exécution et la fenêtre d'administration de 4D Server :

Type de process type	Icone
Process worker préemptif	
Process worker coopératif	

Principes d'utilisation

Le schéma suivant montre une séquence de communication type entre deux workers et un formulaire. Les informations échangées sont de simples chaînes de caractères :



1. Worker_1 appelle Worker_2 et passe "Hello!" en paramètre.
2. Worker_2 récupère et lit le message. Il rappelle Worker_1 et lui passe "Hello!" en paramètre.
3. Worker_2 appelle alors le formulaire MyForm et lui passe "I work" en paramètre. Il démarre une opération longue et son statut devient 'busy' (occupé).
4. Worker_1 appelle Worker_2 deux fois mais, grâce au système de messagerie asynchrone, les messages sont simplement placés dans une file d'attente. Ils sont ensuite traités une fois que Worker_2 est disponible - après que Worker_2 a appelé MyForm.

APPELER PROCESS

APPELER PROCESS (process)

Paramètre	Type		Description
process	Entier long	⇨	Numéro du process

Description

APPELER PROCESS appelle le formulaire affiché dans la fenêtre au premier plan de *process*.

Important : APPELER PROCESS ne fonctionne qu'avec des process tournant sur la même machine.

Si vous appelez un process qui n'existe pas, la commande ne fait rien.

Si *process* (le process appelé) n'a aucune fenêtre ou si aucun formulaire n'est affiché, rien ne se passe. Le formulaire affiché dans le process appelé reçoit un événement Sur appel extérieur. Cet événement doit avoir été sélectionné pour le formulaire dans la fenêtre des **propriétés de formulaire** en mode Développement, et vous devez le traiter dans la méthode formulaire. Si l'événement n'est pas sélectionné ou géré dans la méthode formulaire, la commande ne fait rien.

Note : La réception de l'événement Sur appel extérieur dans un formulaire entrée provoque le changement du contexte de saisie du formulaire. En particulier, si un champ était en cours de modification, l'événement formulaire Sur données modifiées est généré.

Le process appelant (dans lequel la commande **APPELER PROCESS** est exécutée) n'attend pas : **APPELER PROCESS** a un effet immédiat. Il est de votre ressort d'écrire, si nécessaire, une boucle d'attente pour traiter une éventuelle réponse du process appelé à l'aide des variables interprocess ou des variables process (réservées à cette utilisation) pouvant être lues et écrites entre les deux process avec les commandes **LIRE VARIABLE PROCESS** et **ECRIRE VARIABLE PROCESS**.

Si vous voulez établir une communication entre des process qui n'affichent pas de formulaires, utilisez les commandes **LIRE VARIABLE PROCESS** et **ECRIRE VARIABLE PROCESS**.

APPELER PROCESS accepte la syntaxe alternative **APPELER PROCESS(-1)**. Pour ne pas ralentir l'exécution d'une méthode, 4D ne redessine pas les variables interprocess à chaque fois qu'elles sont modifiées. Si vous passez -1 au lieu du numéro du process dans le paramètre *process* de la commande **APPELER PROCESS**, toutes les variables interprocess affichées dans toutes les fenêtres de tous les process seront mises à jour et redessinées.

Exemple

Reportez-vous à l'exemple de la section **Méthode base Sur fermeture**.

APPELER WORKER

APPELER WORKER (process ; méthode {; param}{; param2 ; ... ; paramN})

Paramètre	Type		Description
process	Texte, Entier long	➡	Nom ou numéro du process worker
méthode	Texte	➡	Nom de la méthode projet à appeler
param	Expression	➡	Paramètre(s) passé(s) à la méthode

Description

La commande **APPELER WORKER** crée ou appelle le process worker dont vous avez passé le nom ou le numéro dans *process* et demande l'exécution de la *methode* dans son contexte avec le ou les paramètre(s) optionnel(s) spécifié(s) dans *param*.

La commande **APPELER WORKER** encapsule *param* dans un message qu'elle envoie dans la boîte aux lettres du worker. Pour plus d'informations sur les process workers, reportez-vous à la section **A propos des workers**.

Dans le paramètre *process*, vous passez soit le nom soit le numéro du process worker :

- Si vous passez le numéro d'un process qui n'existe pas, ou si le process spécifié n'a pas été créé par **APPELER WORKER** ni par 4D lui-même (tel quel le process principal de l'application), **APPELER WORKER** ne fait rien.
- Si vous passez le nom d'un process qui n'existe pas, un nouveau process worker est créé.

Note : Le **process principal**, créé par 4D à l'ouverture de la base pour l'interface utilisateur et le mode application, est un process worker et peut être appelé par **APPELER WORKER**. Le nom de ce process pouvant varier en fonction de la langue de 4D, il est préférable de le désigner par son numéro (toujours 1) lorsque vous utilisez **APPELER WORKER**.

Le process worker est affiché dans la liste des process de l'Explorateur d'exécution et est retourné par la commande **INFORMATIONS PROCESS** lorsqu'elle est appliquée à ce process.

Dans le paramètre *méthode*, vous passez le nom de la méthode projet que vous voulez exécuter dans le contexte du process worker. Vous pouvez passer une chaîne vide ; dans ce cas, le worker exécute la méthode utilisée à l'origine pour démarrer son process, s'il y en a (c'est-à-dire, la méthode de démarrage du worker).

Note : Il n'est pas possible de passer une chaîne vide dans *méthode* lorsque la commande appelle le process principal (process n°1), puisqu'il n'est pas démarré via une méthode projet. Par conséquent, ***APPELER WORKER(1;"")*** ne fait rien.

Vous pouvez utiliser le paramètre optionnel *param* pour passer un ou plusieurs paramètres à la *méthode*. Vous passez les paramètres de la mème façon que pour une sous-routine (voir la section **Passer des paramètres aux méthodes**). Au début de son exécution dans le contexte du process, la méthode du process reçoit les valeurs des paramètres dans *$1*, *$2*, etc. Notez qu'un tableau ne peut pas être passé comme paramètre à une méthode. Par ailleurs, dans le contexte de la commande **APPELER WORKER**, les considérations suivantes doivent être prises en compte :

- Les pointeurs vers les tables ou les champs sont autorisés.
- Les pointeurs vers les variables, notamment les variables locales ou les variables process, ne sont pas recommandés car ils peuvent être indéfinis au moment de leur accès par la méthode du process.
- Si vous passez un paramètre de type objet, 4D crée une copie de l'objet dans le process de destination si le worker est exécuté dans un process autre que celui qui appelle la commande **APPELER WORKER**.

Un process worker reste actif jusqu'à la fermeture de l'application ou jusqu'à ce que la commande **TUER WORKER** soit appelée explicitement pour ce process. Pensez à appeler cette commande une fois que le process worker n'est plus utile afin de libérer l'espace mémoire.

Exemple

Dans un formulaire, un bouton démarre un calcul, par exemple des statistiques pour l'année sélectionnée. Ce bouton crée ou appelle un process worker qui effectue le calcul des données tandis que l'utilisateur peut continuer à travailler dans le formulaire.

Voici la méthode du bouton :

```
 //appelle le process worker vNomWorker avec le paramètre
C_ENTIER LONG(vAnnée)
vAnnée:=2015 //peut être sélectionné par l'utilisateur dans le formulaire
APPELER WORKER("monWorker";"méthodeWorker";vAnnée;Fenetre formulaire courant)
```

Voici le code de *méthodeWorker* :

```
 //voici la méthode du worker
 //peut être préemptif ou coopératif
C_ENTIER LONG($1) //reçoit l'année
C_ENTIER LONG($2) //reçoit la référence de la fenêtre
C_OBJET(vRésultatStatistiques) //stockage de résultats statistiques
... //calcul des statistiques
 //une fois le calcul terminé, rappel du formulaire avec les valeurs calculées
 //vRésultatStatistiques peut afficher les résultats dans le formulaire
APPELER FORMULAIRE($2;"affichageStats";vRésultatStatistiques)
```

ECRIRE VARIABLE PROCESS

ECRIRE VARIABLE PROCESS (process ; varDestination ; exprSource {; varDestination2 ; exprSource2 ; ... ; varDestinationN ; exprSourceN})

Paramètre	Type		Description
process	Entier long	➡	Numéro de process de destination
varDestination	Variable	➡	Variable de destination
exprSource	Variable	➡	Expression source (ou variable source)

Description

La commande **ECRIRE VARIABLE PROCESS** écrit la ou les valeur(s) de *exprSource* (*exprSource2*, etc.) dans la ou les variable(s) process *varDestination* (*varDestination2*, etc.) du process de destination dont le numéro est passé dans *process*.

Chaque variable de destination peut être une variable ou un élément de tableau. Tenez cependant compte des restrictions évoquées ci-dessous.

Pour chaque association *varDestination;exprSource*, le type de l'expression doit être compatible avec la variable de destination, sinon vous pourrez obtenir des variables avec des valeurs incorrectes. En mode interprété, si la variable de destination n'existe pas, elle est créée et reçoit l'expression. En mode compilé, si aucune variable n'est associée au process de destination, une erreur est retournée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

Lorsque le process courant écrit les variables du process de destination, ce dernier n'est averti en aucune manière de l'écriture de l'instance de ses variables par un autre process.

4D Server : A partir d'un 4D Client, vous pouvez écrire des variables dans un process de destination exécuté sur le poste serveur (procédure stockée). Pour cela, passez dans process le numéro du process serveur en négatif, c'est-à-dire précédé du signe - (moins).
Attention, la communication process “intermachine” permise par les commandes **ECRIRE VARIABLE PROCESS**, **ECRIRE VARIABLE PROCESS** et **VARIABLE VERS VARIABLE** n’est possible que du client vers le serveur. C’est toujours un process client qui lit ou écrit les variables d’une procédure stockée.

Astuce : Si vous ne connaissez pas le numéro du process serveur de destination, vous pouvez tout de même écrire dans les variables interprocess du serveur. Pour cela, il vous suffit de passer toute valeur négative dans *process*. En d'autres termes, il n'est pas nécessaire de connaître précisément le numéro d'un process exécuté sur le serveur pour utiliser **ECRIRE VARIABLE PROCESS** avec des variables interprocess du serveur.
Cette possibilité s'avère particulièrement utile dans le cas d'une procédure stockée lancée sur le serveur par l'intermédiaire de la **Méthode base Sur démarrage serveur**. Comme les postes clients ne connaissent pas automatiquement le numéro de ce process serveur, il vous suffit de passer une valeur négative (n'importe laquelle) dans le paramètre *process*.

Restrictions

ECRIRE VARIABLE PROCESS n'accepte pas de variables locales comme variables de destination.

ECRIRE VARIABLE PROCESS accepte tout type de variable process ou interprocess de destination, à l'exception :

- des variables de type Pointeur.
- des tableaux de tous types. Pour écrire un tableau entier d'un process vers un autre, utilisez la commande **VARIABLE VERS VARIABLE**. Notez cependant que **ECRIRE VARIABLE PROCESS** vous permet d'écrire des éléments de tableaux.
- des éléments de tableaux de pointeurs et des éléments de tableaux à deux dimensions.

Le process de destination doit être un process utilisateur, ce ne peut être un des process du moteur de 4D. Si le process de destination n'existe pas, la commande ne fait rien.

Exemple 1

La ligne de code suivante affecte une chaîne vide à la variable Texte *vtCurStatus* du process dont le numéro est *$vlProcess* :

```
ECRIRE VARIABLE PROCESS($vlProcess;vtCurStatus;"")
```

Exemple 2

La ligne de code suivante affecte la variable Texte *vtCurStatus* du process dont le numéro est *$vlProcess* à la valeur de la variable *$vtInfo* depuis la méthode en cours d'exécution du process courant :

```
ECRIRE VARIABLE PROCESS($vlProcess;vtCurStatus;$vtInfo)
```

Exemple 3

La ligne de code suivante affecte la variable Texte *vtCurStatus* du process dont le numéro est *$vlProcess* à la valeur de la même variable dans le process courant :

```
ECRIRE VARIABLE PROCESS($vlProcess;vtCurStatus;vtCurStatus)
```

Note : La première *vtCurStatus* désigne l'instance de la variable dans le process de destination, la seconde *vtCurStatus* désigne l'instance de la variable dans le process courant.

Exemple 4

L'exemple suivant place séquentiellement en majuscules les éléments d'un tableau process depuis le process désigné par *$vlProcess*:

```
LIRE VARIABLE PROCESS($vlProcess;vl_IPCom_Array;$vlSize)
Boucle($vlElem;1;$vlSize)
   LIRE VARIABLE PROCESS($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
   ECRIRE VARIABLE PROCESS($vlProcess;at_IPCom_Array{$vlElem};Majusc($vtElem))
Fin de boucle
```

Note : Dans cet exemple, la variable process *vl_IPCom_Array* doit être gérée par les process source/destination et contient la taille du tableau *at_IPCom_Array*.

Exemple 5

L'exemple suivant écrit l'instance des variables *v1*, *v2*, *v3* dans le process de destination à partir de l'instance de ces mêmes variables dans le process courant :

```
ECRIRE VARIABLE PROCESS($vlProcess;v1;v1;v2;v2;v3;v3)
```

EFFACER SEMAPHORE

EFFACER SEMAPHORE (sémaphore)

Paramètre	Type		Description
sémaphore	Chaîne	⇨	Sémaphore à effacer

Description

EFFACER SEMAPHORE permet d'effacer le *sémaphore* précédemment créé par la fonction **Semaphore**.

La règle d'utilisation est que tous les sémaphores doivent être effacés lorsqu'ils ne sont plus nécessaires. Si les sémaphores ne sont pas effacés, ils restent en mémoire jusqu'à la fermeture du process dans lequel ils ont été créés.
Un process ne peut effacer que les sémaphores qu'il a créés. Si vous tentez d'effacer un sémaphore depuis un autre process que celui qui l'a créé, **EFFACER SEMAPHORE** ne fait rien.

Note : Attention, 4D tient compte de la casse des caractères en ce qui concerne les noms de sémaphores (le programme considère par exemple que "MonSémaphore" est différent de "monsémaphore").

Exemple

Reportez-vous à l'exemple de la fonction **Semaphore**.

LIRE VARIABLE PROCESS

LIRE VARIABLE PROCESS (process ; varSource ; varDestination {; varSource2 ; varDestination2 ; ... ; varSourceN ; varDestinationN})

Paramètre	Type		Description
process	Entier long	⇒	Numéro de process source
varSource	Variable	⇒	Variable source
varDestination	Variable	⇐	Variable de destination

Description

La commande **LIRE VARIABLE PROCESS** lit la valeur de la ou des variable(s) process *varSource* (*varSource2*, etc.) depuis le process source dont le numéro est passé dans *process* et la retourne dans la ou les variables(s) *varDestination* (*varDestination2*, etc.) du process courant.

Chaque variable source peut être une variable, un tableau ou un élément de tableau. Tenez cependant compte des restrictions évoquées plus bas.

Pour chaque association *varSource;varDestination* les types des deux variables doivent être compatibles, sinon vous pourrez obtenir des valeurs non significatives.

Le process courant "pille" les variables du process de destination : ce dernier n'est averti en aucune manière de la lecture de l'instance de ses variables par un autre process.

4D Server : A partir d'un 4D Client, vous pouvez lire des variables dans un process de destination exécuté sur le poste serveur (procédure stockée). Pour cela, passez dans *process* le numéro du process serveur en négatif, c'est-à-dire précédé du signe - (moins).
Attention, la communication process “intermachine” permise par les commandes **LIRE VARIABLE PROCESS**, **ECRIRE VARIABLE PROCESS** et **VARIABLE VERS VARIABLE** n’est possible que du client vers le serveur. C’est toujours un process client qui lit ou écrit les variables d’une procédure stockée.

Astuce : Si vous ne connaissez pas le numéro du process serveur source, vous pouvez tout de même lire les variables interprocess du serveur. Pour cela, il vous suffit de passer toute valeur négative dans *process*. En d'autres termes, il n'est pas nécessaire de connaître précisément le numéro d'un process exécuté sur le serveur pour utiliser **LIRE VARIABLE PROCESS** avec les variables interprocess du serveur.
Cette possibilité s'avère particulièrement utile dans le cas d'une procédure stockée lancée sur le serveur par l'intermédiaire de la **Méthode base Sur démarrage serveur**. Comme, par défaut, les postes clients ne connaissent pas le numéro de ce process serveur, il vous suffit de passer une valeur négative (n'importe laquelle) dans le paramètre *process*.

Restrictions

LIRE VARIABLE PROCESS n'accepte pas de variables locales comme variables sources.
En revanche, les variables de destination peuvent être interprocess, process ou locales. Vous pouvez "recevoir" les valeurs uniquement dans des variables, pas dans des champs.
LIRE VARIABLE PROCESS accepte tout type de variable source, process ou interprocess, à l'exception des variables de type :

- Pointeur
- Tableau de pointeurs
- Tableau à deux dimensions

Le process source doit être un process utilisateur, ce ne peut être un des process du moteur de 4D. Si le process source n'existe pas, la commande ne fait rien.

Note : En mode interprété, si une variable source n'existe pas, la valeur indéfinie est retournée. Vous pouvez le détecter en testant la variable de destination correspondante à l'aide de la fonction **Type**. En mode compilé, si aucune variable n'est associée au process source, une erreur est retournée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande **Type**.

Exemple 1

La ligne de code suivante lit la valeur de la variable Texte *vtCurStatus* dans le process dont le numéro est *$vlProcess* et retourne le résultat dans la variable process *vtInfo* du process courant :

```
LIRE VARIABLE PROCESS($vlProcess;vtCurStatus;vtInfo)
```

Exemple 2

La ligne de code suivante fait la même chose mais retourne la valeur dans la variable locale *$vtInfo* de la méthode s'exécutant dans le process courant :

```
LIRE VARIABLE PROCESS($vlProcess;vtCurStatus;$vtInfo)
```

Exemple 3

La ligne de code suivante fait la même chose mais retourne la valeur dans la même variable *vtCurStatus* du process courant :

```
LIRE VARIABLE PROCESS($vlProcess;vtCurStatus;vtCurStatus)
```

Note : La première *vtCurStatus* désigne l'instance de la variable dans le process source, la seconde *vtCurStatus* désigne l'instance de la variable dans le process courant.

Exemple 4

L'exemple suivant lit séquentiellement les éléments d'un tableau process depuis le process indiqué par *$vlProcess* :

```
LIRE VARIABLE PROCESS($vlProcess;vl_IPCom_Array;$vlSize)
Boucle($vlElem;1;$vlSize)
   LIRE VARIABLE PROCESS($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
   ` Faire quelque chose avec $vtElem
Fin de boucle
```

Note : Dans cet exemple, la variable process *vl_IPCom_Array* doit être gérée par le process source et contient la taille du tableau *at_IPCom_Array*.

Exemple 5

L'exemple suivant fait la même chose que le précédent mais lit le tableau dans son intégralité au lieu de le faire élément par élément :

```
LIRE VARIABLE PROCESS($vlProcess;at_IPCom_Array;$anArray)
Boucle($vlElem;1;Taille tableau($anArray))
   ` Faire quelque chose avec $anArray{$vlElem}
Fin de boucle
```

Exemple 6

L'exemple suivant lit l'instance des variables *v1*,*v2*,*v3* dans le process source et retourne leurs valeurs dans l'instance des mêmes variables du process courant :

```
LIRE VARIABLE PROCESS($vlProcess;v1;v1;v2;v2;v3;v3)
```

Exemple 7

Reportez-vous à l'exemple de la commande **PROPRIETES GLISSER DEPOSER**.

Semaphore

Semaphore (sémaphore {; nbTicks}) -> Résultat

Paramètre	Type		Description
sémaphore	Chaîne	⇒	Sémaphore à tester et à positionner
nbTicks	Entier long	⇒	Temps d'attente maximum
Résultat	Booléen	⇐	sémaphore a été correctement créé (Faux) ou sémaphore était déjà créé (Vrai)

Description

Un sémaphore est un drapeau visible par chaque poste client ou chaque process sur un même poste. Un sémaphore a simplement pour rôle d'exister ou de ne pas exister. Chaque méthode exécutée par un utilisateur peut tester la présence d'un sémaphore. Un sémaphore ne peut être effacé que par le poste client ou le process qui l'a créé. En créant et en testant des sémaphores, vous permettez aux méthodes de communiquer entre les postes clients et les process. Les sémaphores ne servent pas à protéger l'accès aux enregistrements — cette gestion est effectuée automatiquement par 4D et 4D Server. Les sémaphores ont pour but d'éviter que plusieurs utilisateurs ou process effectuent la même opération en même temps.

La fonction **Semaphore** retourne Vrai et ne fait rien si *sémaphore* existe. Si *sémaphore* n'existe pas, **Semaphore** le crée et retourne Faux. Un seul utilisateur à la fois peut créer un sémaphore. Si **Semaphore** retourne Faux, cela indique que *sémaphore* n'existait pas, mais cela signifie également que *sémaphore* a été créé et positionné dans le process d'où l'appel a été effectué.

Semaphore retourne Faux si le *sémaphore* n'existait pas. La fonction retourne également Faux si le sémaphore avait été déjà positionné par le process d'où l'appel a été effectué.

Un sémaphore est limité à 255 caractères, métacaractère ($) inclus. Si vous passez une chaîne plus longue, elle est tronquée. Attention, 4D tient compte de la casse des caractères en ce qui concerne les noms de sémaphores (le programme considère par exemple que "MonSémaphore" est différent de "monsémaphore").

Le paramètre optionnel *nbTicks* vous permet de spécifier un délai d'attente en ticks (1 tick = 1/60ème de seconde) si *sémaphore* est déjà positionné. Dans ce cas, avant de retourner Vrai, la fonction attend, dans la limite du temps fixé, que *sémaphore* se libère (auquel cas elle retourne Faux). Si le délai expire sans que *sémaphore* ait été libéré, **Semaphore** retourne Vrai.

Il y a deux types de sémaphores dans 4D : les sémaphores locaux et les sémaphores globaux.

- Un sémaphore local est visible par tous les process d'un même poste et seulement sur ce poste. Vous déclarez un sémaphore local en préfixant son nom avec le signe dollar ($). Les sémaphores locaux permettent de contrôler des opérations entre les différents process exécutés sur le même poste. Par exemple, un sémaphore local peut être utilisé pour gérer les accès à un tableau interprocess appelé par tous les process d'une base de données mono-utilisateur ou d'un poste client.
- Un sémaphore global est visible par tous les utilisateurs et tous les process. Les sémaphores globaux permettent de contrôler des opérations entre les postes clients d'une base multi-utilisateurs.

Le principe de fonctionnement des sémaphores globaux et locaux est identique. Leur différence réside uniquement dans leur portée, c'est-à-dire leur visibilité. En client-serveur, les sémaphores globaux sont visibles pour tous les process de tous les postes clients et du serveur. Un sémaphore local n'est visible que pour les process du poste sur lequel il a été créé.

Avec 4D, les sémaphores globaux et locaux ont la même portée car il n'y a qu'un seul utilisateur. Cependant, si votre base est utilisée dans les deux environnements, n'hésitez pas à employer des sémaphores globaux et locaux, en fonction de vos besoins.

Note : Les sémaphores locaux sont recommandés lorsque l'usage d'un sémaphore est nécessaire pour gérer un aspect local à un client de l'application, comme par exemple l'interface ou un tableau de valeurs interprocess. L'utilisation d'un sémaphore global provoquerait dans ce cas non seulement des échanges réseau inutiles, mais en plus pourrait affecter inutilement d'autres postes clients. Le sémaphore local évitera ces effets indésirables.

Exemple 1

Le code type d'utilisation d'un sémaphore est le suivant :

```
Tant que(Semaphore("MonSemaphore";300))
   APPELER 4D
Fin tant que
 // placer ici le code protégé par le sémaphore
EFFACER SEMAPHORE("MonSemaphore")
```

Exemple 2

Dans l'exemple suivant, vous souhaitez empêcher que deux utilisateurs effectuent simultanément une mise à jour globale des prix dans une table [Produits]. Pour cela, des sémaphores sont utilisés :

```
Si(Semaphore("MAJPrix")) ` Essai de création du sémaphore
   ALERTE("Un autre utilisateur est déjà en train de mettre à jour les prix. Essayez plus tard.")
Sinon
   MAJdesPrix ` Méthode de mise à jour des prix
   EFFACER SEMAPHORE("MAJPrix")) ` Effacer le sémaphore
Fin de si
```

Exemple 3

L'exemple suivant illustre l'utilisation d'un sémaphore local. Dans une base comportant plusieurs process, vous souhaitez maintenir une liste de "Choses à faire". Vous envisagez de la maintenir à jour dans un tableau interprocess et non dans une table. Vous devez empêcher les accès simultanés à l'aide d'un sémaphore. Dans ce cas, il vous suffit d'utiliser un sémaphore local car la liste "Choses à faire" est pour votre utilisation personnelle.

Le tableau interprocess est initialisé dans la méthode base **Sur ouverture** :

```
TABLEAU TEXTE(<>ListeAFaire;0) ` La liste de choses à faire est vide
```

Voici la méthode utilisée pour ajouter des éléments à la "liste des choses à faire" :

```
 ` Méthode projet AJOUTER LISTE A FAIRE
 ` AJOUTER LISTE A FAIRE (Texte)
 ` AJOUTER LISTE A FAIRE (Elément la liste à faire)

C_TEXTE($1) ` Paramètre passé à la commande
Si(Non(Semaphore("$AccèsListe";300))) ` Attendre 5 secondes si un sémaphore existe déjà
   $vlElem:=Taille tableau(<>ListeAFaire)+1
   INSERER DANS TABLEAU(<>ListeAFaire;$vlElem)
   <>ListeAFaire{$vlElem}:=$1
   EFFACER SEMAPHORE("$AccèsListe") ` Effacer le sémaphore
Fin de si
```

Vous pouvez appeler cette méthode depuis n'importe quel process.

Exemple 4

Cette méthode permet de ne pas exécuter une méthode si le sémaphore est posé ; la méthode informe la méthode d'appel avec un code d'erreur et un texte en clair.

Syntaxe :

```
$L_Erreur:=Semaphore_proof(->$T_Text_error)
```

```
 // Structure de protection par sémaphore
C_ENTIER LONG($0)
C_POINTEUR($1) // message d'erreur

  // Début de la méthode
C_ENTIER LONG($L_MyError)
$L_MyError:=1

C_TEXTE($T_Sema_local)
$T_Sema_local:="$tictac"

Si(Semaphore($T_Sema_local;300))
     // On a attendu 300 ticks mais le sémaphore
     // n'a pas été libéré par celui qui l'avait posé :
     // on arrive ici
   $L_MyError:=-1

Sinon

       // Cette méthode n'est exécutée que par un process à la fois

       // Nous avons posé le sémaphore en même temps que nous entrions
       // il n'y a que nous qui pouvons le supprimer

       // Faire quelque chose
         ...
       // Finir en effaçant le sémaphore
   EFFACER SEMAPHORE($T_Sema_local)
Fin de si

C_TEXTE($T_Message)
Si($L_MyError=-1)
   $T_Message:="Le sémaphore "+$T_Sema_local+" a bloqué l'accès à la suite du code"
Sinon
   $T_Message:="OK"
Fin de si

$0:=$L_MyError
$1->:=$T_Message  // la méthode d'appel reçoit un code d'erreur et une explication en clair
```

Tester semaphore

Tester semaphore (sémaphore) -> Résultat

Paramètre	Type		Description
sémaphore	Chaîne	→	Nom du sémaphore à tester
Résultat	Booléen	←	Vrai = le sémaphore existe, Faux = le sémaphore n'existe pas

Description

La commande **Tester semaphore** permet de tester l'existence d'un sémaphore.

A la différence de la fonction **Semaphore**, **Tester semaphore** ne crée pas le sémaphore s'il n'existe pas.
Si le sémaphore existe, la fonction retourne **Vrai**, s'il n'existe pas elle retourne **Faux**.

Note : Attention, 4D tient compte de la casse des caractères en ce qui concerne les noms de sémaphores (le programme considère par exemple que "MonSémaphore" est différent de "monsémaphore").

Exemple

Cet exemple permet de connaître l'état d'un traitement (en l'occurrence, la modification d'un code) sans modifier le sémaphore :

```
Creer fenetre(x1;x2;y1;y2;-Fenêtre palette)
Repeter
   Si(Tester semaphore("Code d'encryptage"))
      POSITION MESSAGE($x3;$y3)
      MESSAGE("Code d'encryptage en cours de modification.")
   Sinon
      POSITION MESSAGE($x3;$y3)
      MESSAGE("Modification du code d'encryptage autorisée.")
   Fin de si
Jusque(StopInfo)
FERMER FENETRE
```

TUER WORKER

TUER WORKER {(process)}

Paramètre	Type		Description
process	Texte, Entier long	⇨	Nom ou numéro du process worker à tuer (process courant si omis)

Description

La commande **TUER WORKER** envoie un message au process worker dont vous avez passé le nom ou le numéro dans *process*, lui demandant d'ignorer tous les messages en attente (s'il y a) et de terminer son exécution à l'issue de la tâche en cours.

Cette commande ne peut être utilisée qu'avec des process workers. Pour plus d'informations, reportez-vous à la section **A propos des workers**.

Dans le paramètre *process*, vous pouvez passer soit le nom soit le numéro du process worker que vous voulez tuer. Si aucun process worker avec le nom ou le numéro spécifié existe, **TUER WORKER** ne fait rien.

Lorsque le paramètre *process* est omis, **TUER WORKER** s'applique au process worker courant et équivaut donc à ***TUER WORKER****(Numero du process courant)*.

Lorsque la commande est appliquée à un worker qui n'a pas été créé explicitement par la commande **APPELER WORKER** (par exemple, le process worker principal de l'application), elle vide uniquement la boîte aux lettres du worker. Par conséquent, ***TUER WORKER****(1)* ne fait rien.

Exemple

Le code suivant (exécuté depuis un formulaire, par exemple) déclenche l'arrêt d'un process worker :

```
APPELER WORKER(vNomWorker;"leWorker";"fin")
```

Dans la méthode du process worker (*leWorker*), vous ajoutez du code pour gérer cette situation :

```
 //méthode leWorker
C_TEXTE($1) //paramètre

Au cas ou
   :($1="appel") //on appelle le worker
      ... //faire quelque chose
   :($1="fin") //on demande au worker de terminer son exécution
      TUER WORKER
Fin de cas
```

VARIABLE VERS VARIABLE

VARIABLE VERS VARIABLE (process ; varDestination ; varSource {; varDestination2 ; varSource2 ; ... ; varDestinationN ; varSourceN})

Paramètre	Type		Description
process	Entier long	➡	Numéro du process de destination
varDestination	Variable	➡	Variable de destination
varSource	Variable	➡	Variable source

Description

La commande **VARIABLE VERS VARIABLE** écrit la valeur de la ou des variable(s) *varSource1* (*varSource2*, etc.), dans la ou les variable(s) process *varDestination* (*varDestination2*, etc.) du process de destination dont vous avez passé le numéro dans *process*.

VARIABLE VERS VARIABLE a un fonctionnement semblable à celui de la commande **ECRIRE VARIABLE PROCESS**, avec cependant les différences suivantes :

- Alors que vous passez comme source à **ECRIRE VARIABLE PROCESS**des expressions (et donc vous ne pouvez pas passer un tableau en totalité), vous devez passer comme source à **VARIABLE VERS VARIABLE** uniquement des variables (et donc vous pouvez passer un tableau en totalité).
- Avec **ECRIRE VARIABLE PROCESS**, chaque variable de destination peut être une variable ou un élément de tableau, mais ne peut pas être un tableau. Avec **VARIABLE VERS VARIABLE**, chaque variable de destination peut être une variable, un tableau ou un élément de tableau.

4D Server : La communication process “intermachine” permise par les commandes **VARIABLE VERS VARIABLE**, **ECRIRE VARIABLE PROCESS** et **LIRE VARIABLE PROCESS** n’est possible que du client vers le serveur. C’est toujours un process client qui lit ou écrit les variables d’une procédure stockée.

Pour chaque association *varDestination;varSource*, le type de la variable source doit être compatible avec la variable de destination, sinon vous pourrez obtenir des variables avec des valeurs non significatives. En mode interprété, si la variable de destination n'existe pas, elle est créée puis le type et la valeur de la variable source lui sont affectés.

Lorsque le process courant écrit les variables du process de destination, ce dernier n'est averti en aucune manière de l'écriture de l'instance de ses variables par un autre process.

Restrictions

VARIABLE VERS VARIABLE n'accepte pas de variables locales comme variables de destination.

VARIABLE VERS VARIABLE accepte tout type de variable process ou interprocess de destination, à l'exception de variables de type :

- Pointeur
- Tableau de pointeurs
- Tableau à deux dimensions

Le process de destination doit être un process utilisateur, ce ne peut être un des process du moteur de 4D. Si le process de destination n'existe pas, une erreur est retournée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

Exemple

L'exemple suivant récupère un tableau process depuis le process désigné par *$vlProcess*, passe séquentiellement tous ses éléments en caractères majuscules puis réécrit entièrement le tableau :

```
LIRE VARIABLE PROCESS($vlProcess;at_IPCom_Tab;$anTab)
Boucle($vlElem;1;Taille tableau($anTab))
   $anTab{$vlElem}:=Majusc($anTab{$vlElem})
Fin de boucle
VARIABLE VERS VARIABLE($vlProcess;at_IPCom_Tab;$anTab)
```

Process (Interface utilisateur)

CACHER PROCESS
MONTRER PROCESS
PASSER AU PREMIER PLAN
Process de premier plan

CACHER PROCESS

CACHER PROCESS (process)

Paramètre	Type		Description
process	Entier long	⇨	Numéro du process à cacher

Description

CACHER PROCESS masque toutes les fenêtres appartenant au process dont le numéro est *process*. Tous les éléments d'interface de *process* sont cachés jusqu'au **MONTRER PROCESS** suivant. La barre de menus du process est aussi cachée. L'ouverture d'une fenêtre alors que le process est caché ne provoquera aucun redessinement d'écran. Si le process est déjà caché, cette commande ne fait rien.

La seule exception à cette règle est la fenêtre du débogueur. Si la fenêtre du débogueur est affichée lorsque *process* est caché, *process* est affiché et passe au premier plan.

Si vous ne voulez pas qu'un process soit affiché lorsqu'il est créé, **CACHER PROCESS** doit être la première commande à appeler dans la méthode du process. Les process Process principal et Gestionnaire du cache ne peuvent pas être cachés à l'aide de cette commande.

Lorsqu'un process est caché, il est toujours en cours d'exécution.

Si vous souhaitez ne cacher qu'une fenêtre du process, utilisez la commande **CACHER FENETRE**.

Exemple

L'exemple suivant cachera toutes les fenêtres appartenant au process courant :

```
CACHER PROCESS(Numero du process courant)
```

MONTRER PROCESS

MONTRER PROCESS (process)

Paramètre	Type		Description
process	Entier long	⇨	Numéro du process dont les fenêtres doivent être affichées

Description

MONTRER PROCESS fait apparaître l'ensemble des fenêtres appartenant à *process*. Cette commande ne passe pas les fenêtres de *process* au premier plan, utilisez pour cela la commande **PASSER AU PREMIER PLAN**.

Si les fenêtres de *process* sont déjà affichées, cette commande ne fait rien.

Exemple

L'exemple suivant affiche le process "Clients", s'il était caché auparavant. Le numéro de process est stocké dans la variable interprocess *<>Clients* :

```
MONTRER PROCESS(<>Clients)
```

PASSER AU PREMIER PLAN

PASSER AU PREMIER PLAN (process)

Paramètre	Type		Description
process	Entier long	⇨	Numéro du process à passer au premier plan

Description

PASSER AU PREMIER PLAN passe les fenêtres du process de numéro *process* au premier plan. Toutes les fenêtres appartenant à *process* passent au premier plan. Si le process est déjà au premier plan, la commande ne fait rien. Si le process est caché, il faut utiliser la commande **MONTRER PROCESS** pour faire d'abord apparaître le process, sinon **PASSER AU PREMIER PLAN** ne fait rien.

Le Process principal et le Process de structure peuvent être passés au premier plan à l'aide de cette commande.

Note : Si le process contient plusieurs fenêtres et que vous souhaitez passer au premier plan une fenêtre spécifique, il est préférable d'utiliser par exemple la commande **CHANGER COORDONNEES FENETRE**.

Exemple

L'exemple suivant est une méthode qui peut être exécutée à partir d'une commande de menu. Elle vérifie si le process au premier plan est le process <>Clients. Sinon, ce process passe au premier plan :

```
Si(Process de premier plan#<>Clients) ` Si la liste des clients n'est pas affichée
   PASSER AU PREMIER PLAN(<>Clients) ` Passer cette liste au premier plan
Fin de si
```

Process de premier plan

Process de premier plan {(*)} -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Numéro du process de la première fenêtre non-flottante
Résultat	Entier	⇐	Numéro du process dont la ou les fenêtre(s) est (sont) au premier plan

Description

Process de premier plan retourne le numéro du process dont la ou les fenêtre(s) est (sont) au premier plan.

Lorsqu'une ou plusieurs fenêtres flottantes sont ouvertes, deux niveaux différents de fenêtres sont distingués :

- les fenêtres standard
- les fenêtres flottantes

Si la fonction **Process de premier plan** est utilisée dans la méthode formulaire ou dans une méthode objet d'une fenêtre flottante, la fonction retourne le numéro du process de la fenêtre flottante au premier plan parmi les fenêtres flottantes. Si vous passez le paramètre optionnel astérisque, la fonction retourne le numéro du process dont la fenêtre est au premier plan, **exception faite du niveau des fenêtres flottantes**.

Exemple

Référez-vous à l'exemple de **PASSER AU PREMIER PLAN**.

Protocole sécurisé

GENERER CLES CRYPTAGE
GENERER DEMANDE CERTIFICAT

GENERER CLES CRYPTAGE

GENERER CLES CRYPTAGE (cléPrivée ; cléPublique {; longueur})

Paramètre	Type		Description
cléPrivée	BLOB	⇐	BLOB devant recevoir la clé privée
cléPublique	BLOB	⇐	BLOB devant recevoir la clé publique
longueur	Entier long	⇒	Longueur des clés en bits [386...2048] Par défaut = 512

Description

La commande **GENERER CLES CRYPTAGE** génère une nouvelle paire de clés RSA. Ces clés, destinées au cryptage/décryptage des données, sont à la base du système de sécurisation des données proposé par 4D. Elles pourront être utilisées, dans le cadre du protocole TLS/SSL, avec le serveur Web 4D (cryptage et sécurité des communications) mais également dans toute base de données (cryptage de données).

Après l'exécution de la commande, les BLOB passés dans les paramètres *cléPrivée* et *cléPublique* contiennent une nouvelle paire de clés de cryptage.

Le paramètre optionnel *longueur* vous permet de préciser la taille (en bits) des clés que vous souhaitez obtenir. Plus une clé est longue, plus son décryptage "frauduleux" sera difficile.
En contrepartie, plus les clés sont longues, plus les délais d'exécution ou de réponse seront importants, en particulier dans le cadre d'une connexion sécurisée.
Par défaut (si vous omettez le paramètre *longueur*), la taille des clés générée est de 512 bits. Vous pouvez générer des clés de 2048 bits, ce qui renforce la sécurité du cryptage, mais ralentira les connexions de votre application Web. Pour augmenter encore la sécurité, vous pouvez envisager de changer de paire de clés assez fréquemment, par exemple tous les six mois.

Les clés générées par cette commande sont au format standard PKCS encodé en base64, ce qui signifie que leur contenu peut être copié et collé dans un e-mail en toute sécurité et sans risque d'altération. Une fois que vous avez obtenu une paire de clés, vous pouvez générer un document texte au format PEM (par exemple à l'aide de la commande **BLOB VERS DOCUMENT**) et stocker les clés dans un endroit sûr.

Important : La clé privée ne doit jamais être diffusée, sous quelque forme que ce soit.

RSA, clés privées et clés publiques

L'algorithme de cryptage RSA employé par la commande **GENERER CLES CRYPTAGE** est basé sur un système de cryptage à double clé : une clé privée et une clé publique. Comme son nom l'indique, la clé publique peut être diffusée auprès de tiers, et permet le décryptage des informations. Il lui correspond une clé privée unique, utilisée pour crypter les données. La clé privée sert au cryptage ; la clé publique, au décryptage (ou inversement). Ce qui est crypté avec une clé ne peut être décrypté qu'avec l'autre.
Les fonctions de cryptage du protocole TLS/SSL sont basées sur ce principe, la clé publique étant incluse dans le certificat envoyé aux navigateurs (cf. section **Utiliser le protocole TLS**).

Ce mode de cryptage est également utilisé par la première syntaxe des commande **CRYPTER BLOB** et **DECRYPTER BLOB**. Ce principe requiert que la clé publique soit diffusée de manière confidentielle.
Il est possible de mêler les clés publiques et privées de deux intervenants pour crypter des données de telle manière que seul le récepteur peut décrypter les données, et seul l'émetteur peut les avoir cryptées. C'est le principe de la seconde syntaxe des commandes **CRYPTER BLOB** et **DECRYPTER BLOB**.

Exemple

Reportez-vous à l'exemple de la commande **CRYPTER BLOB**.

GENERER DEMANDE CERTIFICAT

GENERER DEMANDE CERTIFICAT (cléPrivée ; demCertif ; tabCodes ; tabLibellés)

Paramètre	Type		Description
cléPrivée	BLOB	⇒	BLOB contenant la clé privée
demCertif	BLOB	⇐	BLOB devant recevoir la demande de certificat
tabCodes	Tableau entier long	⇒	Liste des codes d'informations
tabLibellés	Tableau chaîne	⇒	Liste des libellés d'informations

Description

La commande **GENERER DEMANDE CERTIFICAT** permet de générer une demande de certificat au format PKCS, directement exploitable par des autorités de certification telles que Verisign® ou Thawthe®. Le certificat est une pièce essentielle du fonctionnement du protocole SSL dans le cadre d'un serveur Web. Il est envoyé à chaque browser se connectant en mode SSL. Il contient la "carte d'identité" du site Web (reprenant les informations que vous saisissez dans la commande), ainsi que sa clé publique — permettant aux browsers de décrypter les informations reçues. En outre, le certificat contient diverses informations ajoutées par l'autorité de certification.

Note : Pour plus d'informations sur le fonctionnement du protocole SSL avec le serveur Web 4D, reportez-vous à la section **Utiliser le protocole TLS**.

La demande de certificat nécessite une paire de clés générée à l'aide de la commande **GENERER CLES CRYPTAGE** et contient diverses informations. C'est en combinant cette demande avec d'autres paramètres qui lui sont propres, que l'autorité de certification sera en mesure de générer un certificat.

Passez dans *cléPrivée* un BLOB contenant la clé privée générée avec la commande **GENERER CLES CRYPTAGE**.

Passez dans *demCertif* un BLOB vide. Après l'exécution de la commande, il contiendra la demande de certificat au format PKCS encodé en base64. Vous pouvez stocker directement ce contenu dans un fichier texte suffixé .pem, par exemple à l'aide de la commande **BLOB VERS DOCUMENT**, pour la faire parvenir à l'autorité de certification.

Important : La clé privée est utilisée pour générer la demande de certificat mais ne doit pas être envoyée à l'autorité de certification.

Vous devez remplir les tableaux *tabCodes* (de type entier long) et *tabLibellés* (de type alpha) avec, respectivement, les numéros de code et les libellés des informations destinées à l'autorité de certification.
Les codes et les libellés attendus peuvent varier en fonction de l'autorité de certification et du mode d'utilisation du certificat. Toutefois, dans le cadre d'une utilisation standard du certificat (connexions d'un serveur Web via SSL), les tableaux doivent contenir les éléments suivants :

Informations à fournir	tabCodes	tabLibellés (Exemples)
CommonName : Nom du domaine	13	www.4D.fr
CountryName : Code du pays (deux lettres)	14	FR
LocalityName : Ville	15	Clichy
StateOrProvinceName : Département, Etat...	16	Hauts de Seine
OrganizationName : Raison sociale	17	4D
OrganizationUnit : Service/Personne en charge du serveur	18	Web Administrator

L'ordre dans lequel les codes et les informations sont insérés dans les tableaux n'a pas d'importance, en revanche les deux tableaux doivent être "synchronisés" : si l'élément {3} du tableau *tabCodes* contient la valeur 15 (nom de la ville), l'élément {3} du tableau *tabLibellés* doit contenir cette information, dans notre exemple Clichy.

Exemple

Un formulaire "Demande de certificat" comporte les six champs nécessaires à l'établissement d'une demande de certificat standard. Le bouton **Générer** crée un document sur disque contenant la demande de certificat. Le document "Cléprivée.txt" contient la clé privée (générée à l'aide la commande **GENERER CLES CRYPTAGE**) doit déjà être présent sur le disque.


```
 // Méthode objet du bouton bGénérer
C_BLOB($vbcléPrivée;$vbDemandeCert)
C_ENTIER LONG($NumTable)
TABLEAU ENTIER LONG($tLCodes;6)
TABLEAU ALPHA(80;$tAInfos;6)

$NumTable:=Table(Table du formulaire courant)
Boucle($i;1;6)
   $tAInfos{$i}:=Champ($NumTable;$i)->
   $tLCodes{$i}:=$i+12
Fin de boucle
Si(Chercher dans tableau($tAInfos;"")#-1)
```

```
      ALERTE("Vous devez remplir tous les champs.")
   Sinon
      ALERTE("Sélectionnez votre clé privée.")
      $vhRefDoc:=Ouvrir document("")
      Si(OK=1)
         FERMER DOCUMENT($vhRefDoc)
         DOCUMENT VERS BLOB(Document;$vbcléPrivée)
         GENERER DEMANDE CERTIFICAT($vbcléPrivée;$vbDemandeCert;$tLCodes;$tAInfos)
         BLOB VERS DOCUMENT("Demande.txt";$vbDemandeCert)
      Sinon
         ALERTE("Clé privée invalide.")
      Fin de si
   Fin de si
```

Recherches et tris

- CHERCHER
- CHERCHER DANS SELECTION
- CHERCHER PAR ATTRIBUT
- CHERCHER PAR ATTRIBUT DANS SELECTION Nouveauté 16.0
- CHERCHER PAR EXEMPLE
- CHERCHER PAR FORMULE
- CHERCHER PAR FORMULE DANS SELECTION
- CHERCHER PAR TABLEAU
- CHERCHER PAR TABLEAU DANS SELECTION
- DECRIRE EXECUTION RECHERCHE
- FIXER DESTINATION RECHERCHE
- FIXER LIMITE RECHERCHE
- FIXER RECHERCHE ET VERROUILLAGE
- Lire dernier chemin recherche
- Lire dernier plan recherche
- LIRE DESTINATION RECHERCHE
- Lire limite recherche
- TRIER
- TRIER PAR FORMULE
- Trouver dans champ

CHERCHER

CHERCHER ({laTable }{;}{ critère {; *}})

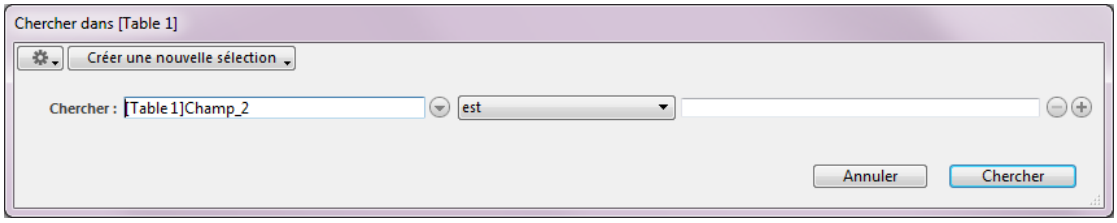
Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle la sélection est créée ou Table par défaut si ce paramètre est omis
critère	Expression	⇒	Critère de recherche
*	Opérateur	⇒	Attente d'exécution de la recherche

Description

La commande **CHERCHER** recherche les enregistrements répondant au(x) critère(s) de recherche spécifié(s) dans *critère* et retourne une sélection d'enregistrements de *laTable*. **CHERCHER** modifie la sélection courante de *laTable* pour le process courant. Le premier enregistrement de la nouvelle sélection devient l'enregistrement courant.

Si vous omettez le paramètre *laTable*, la commande s'applique à la table par défaut. Si aucune table par défaut n'a été définie, une erreur est générée.

Si vous ne passez ni le paramètre *critère* ni le paramètre *, **CHERCHER** affiche la boîte de dialogue de l'Editeur de recherches de 4D pour *table* (sauf lorsqu'il s'agit de la dernière ligne d'une recherche complexe, cf. ci-dessous) :



Pour plus d'informations sur l'utilisation de cet éditeur, reportez-vous au manuel *Mode Développement*.
L'utilisateur construit la recherche puis clique sur le bouton **Chercher** ou **Chercher dans sélection**. Si la recherche est correctement effectuée et n'est pas interrompue, la variable système OK prend la valeur *1*. Si l'utilisateur clique sur **Annuler**, la commande **CHERCHER** est interrompue sans effectuer de recherche et la variable OK prend la valeur *0* (zéro).

Exemple 1

L'exemple suivant affiche l'Editeur de recherches pour la table [Produits] :

```
CHERCHER([Produits])
```

Exemple 2

L'exemple suivant affiche l'Editeur de recherches pour la table par défaut (si elle a été définie) :

```
CHERCHER
```

Si vous spécifiez le paramètre *critère*, l'Editeur de recherches ne s'affiche pas et la recherche est entièrement définie par programmation. Pour des recherches simples (recherches sur un seul champ), vous appelez **CHERCHER** une seule fois avec le paramètre *critère* construit de la manière décrite plus bas. Pour des recherches complexes (recherches sur de multiples champs ou avec de multiples conditions), vous appelez **CHERCHER** autant de fois que nécessaire avec le paramètre *critère* et le paramètre optionnel * sauf pour la dernière ligne **CHERCHER** (qui déclenche la recherche).

Exemple 3

L'exemple suivant recherche les [Personnes] dont le nom commence par "a" :

```
CHERCHER([Personnes];[Personnes]Nom="a@")
```

Exemple 4

L'exemple suivant recherche les [Personnes] dont le nom commence par "a" ou "b" :

```
CHERCHER([Personnes];[Personnes]Nom="a@";*) ` * indique qu'il y a un autre critère de recherche
CHERCHER([Personnes];|;[Personnes]Nom="b@")
  ` Pas de * : indique la fin de la définition des critères et lance l'exécution de la recherche
```

Note : Le mode d'interprétation du caractère @ dans les recherches peut être modifié via une option des préférences. Pour plus d'informations, reportez-vous à la section **Opérateurs de comparaison**.

Construction d'une ligne de recherche

Le paramètre *critère* utilise la syntaxe suivante :
{opérateur ; } champ comparateur valeur

- L'opérateur est utilisé pour lier deux appels à **CHERCHER** lors d'une définition de recherche complexe. Les opérateurs disponibles sont les mêmes

que ceux proposés dans l'Editeur de recherches :

Opérateur	Symbole
ET	&
OU	|
Sauf	#

L'opérateur est optionnel et n'est pas nécessaire pour le premier appel à **CHERCHER** lors d'une recherche complexe. Il est également inutile si votre recherche s'écrit sur une seule ligne. Si vous l'omettez à l'intérieur d'une recherche complexe, le **ET** (&) est utilisé par défaut.

- Le *champ* est le champ sur lequel va porter la recherche. Il peut provenir d'une autre table si celle-ci est la table 1 d'une table liée à *table* par un lien automatique ou manuel.
- Le *comparateur* est l'élément qui va permettre de confronter *champ* et *critèreRecherche*. Voici la liste des comparateurs possibles :

Comparateur	Symbole à utiliser avec CHERCHER
Egal à	=
Différent de	#
Inférieur à	<
Supérieur à	>
Inférieur ou égal à	<=
Supérieur ou égal à	>=
Contient mot-clé	%

Note : Il est possible de définir le comparateur sous la forme d'une expression alphanumérique au lieu d'un symbole. Dans ce cas, il est obligatoire d'utiliser des points-virgules pour dissocier les éléments de la chaîne de recherche. Ce principe permet par exemple de créer des séquences de recherches paramétrables en faisant varier le comparateur, ou de construire des interfaces de recherche utilisateur personnalisées. Reportez-vous à l'exemple 19.
Exemple 16

- La *valeur* représente ce qui va être comparé au contenu de *champ*. La valeur peut être toute expression du même type que *champ*. Le type de la valeur n'est évalué qu'une seule fois, au démarrage de la recherche, et ne l'est donc pas pour chaque enregistrement. Si la recherche porte sur le contenu d'une chaîne de caractères, utilisez dans 'valeur' le symbole "@" pour isoler le contenu à rechercher, par exemple "@Dupon@". Il est à noter, dans ce cas, que la recherche ne tire que partiellement parti de l'index (compacité du stockage des données).
  La recherche par mots-clés n'est disponible qu'avec des champs de type alpha ou texte. Pour plus d'informations sur ce type de recherche, reportez-vous à la section **Opérateurs de comparaison**.

Voici les règles à observer pour la construction de séquences de recherche :

- La première ligne ne doit pas contenir d'opérateur.
- Les suivantes peuvent débuter par un opérateur de liaison. Si vous l'omettez, l'opérateur **ET** (&) est utilisé par défaut.
- Toutes les lignes, à l'exception de la dernière, doivent s'achever par le symbole *.
- Pour lancer la recherche, ne passez pas le paramètre * lors de la construction de votre dernière ligne. Autre solution : vous pouvez exécuter la commande **CHERCHER** sans autre paramètre que la table (l'Editeur de recherches ne s'affiche pas ; au lieu de cela, les lignes de recherche complexes définies auparavant sont exécutées).

Note : Chaque table maintient sa propre construction de recherche courante. Cela signifie que vous pouvez créer de multiples recherches simultanément, une pour chaque table. Dans ce cas, vous devez passer le paramètre *table* ou spécifier une table par défaut.

Quelle que soit la manière dont la recherche a été définie :

- Si l'exécution d'une commande **CHERCHER** nécessite un certain temps, 4D affiche automatiquement un message contenant un thermomètre de progression. Ces thermomètres peuvent être cachés à l'aide des commandes **LAISSER MESSAGES** et **SUPPRIMER MESSAGES**. Si le thermomètre de progression est affiché, l'utilisateur peut cliquer sur le bouton **Stop** pour interrompre l'opération. Si la recherche s'est correctement déroulée, la variable système OK prend la valeur 1. Sinon, si la recherche est interrompue, OK prend la valeur 0 (zéro).
- Si des champs indexés sont spécifiés, la recherche est optimisée à chaque fois que c'est possible (la recherche commence par les champs indexés), réduisant au maximum la durée de l'opération. La commande tire parti des index composites pour les recherches utilisant le **ET** (&).

Exemple 5

Nous recherchons tous les enregistrements dont le nom correspond à "Dupont" :

```
CHERCHER([Personnes];[Personnes]Nom="Dupont")
```

Note : Si le champ Nom est indexé, nous bénéficions donc d'une recherche accélérée tirant parti de l'index.

Rappel : Cette recherche trouvera les enregistrements tels que "Dupont", "dupont", "DUPONT", etc. Si vous voulez que la recherche tienne compte des majuscules/minuscules, définissez des critères supplémentaires utilisant les codes de caractères.

Exemple 6

Nous recherchons les personnes se nommant "Dupont" et se prénommant "Jean". Le champ Nom est indexé. En revanche, le champ Prénom ne l'est pas :

```
CHERCHER([Personnes];[Personnes]Nom="Dupont";*) ` Chercher toute personne qui s'appelle Dupont
CHERCHER([Personnes];&;[Personnes]Prénom="Jean") ` dont le prénom est Jean
```

Cet exemple effectue dans un premier temps une recherche rapide sur le champ indexé Nom, ce qui réduit la sélection d'enregistrements à ceux des personnes s'appelant Dupont. La recherche s'effectue ensuite séquentiellement sur le champ Prénom, mais nous serons peu pénalisés puisqu'elle s'exécute parmi une présélection d'enregistrements.

Note : Cette recherche est particulièrement optimisée si la base contient un index composite incluant les champs *[Personnes]Nom+[Personnes]Prénom*. Dans ce cas, la commande tire parti de l'index et la recherche est entièrement indexée.

Exemple 7

L'exemple suivant recherche les personnes se nommant Dupont ou Blanc. Le champ Nom est indexé :

```
 ` Chercher toute personne qui s'appelle Dupont…
CHERCHER([Personnes];[Personnes]Nom="Dupont";*)
CHERCHER([Personnes];|;[Personnes]Nom="Blanc") ` ou Blanc
```

La commande utilise l'index du champ Nom pour les deux recherches. Les deux recherches sont effectuées, et leurs résultats sont placés dans des ensembles internes qui sont finalement combinés par l'intermédiaire d'une opération Union.

Exemple 8

L'exemple suivant recherche des personnes qui ne travaillent pas pour une société. La recherche est effectuée en testant si le nom de la société est une chaîne vide.

```
CHERCHER([Personnes];[Personnes]Société="") ` Chercher les personnes sans société
```

Exemple 9

L'exemple suivant recherche chaque personne se nommant "Dupont" et travaillant dans une société basée à Paris. La deuxième recherche utilise un champ venant d'une autre table. Cette recherche peut être effectuée parce que la table [Personnes] est liée à la table [Société] par un lien de N vers 1 :

```
CHERCHER([Personnes];[Personnes]Nom="Dupont";*) ` Chercher toute personne qui s'appelle Dupont…
CHERCHER([Personnes];&;[Société]Ville ="Paris") ` ...qui travaille pour une société à Paris
```

Exemple 10

L'exemple suivant recherche l'enregistrement de chaque personne dont l'initiale du nom est située entre les lettre A (incluse) et M (incluse) :

```
CHERCHER([Personnes];[Personnes]Nom<"n") ` Trouver toute personne entre A et M
```

Exemple 11

L'exemple suivant recherche les enregistrements des personnes habitant soit Paris soit Lyon :

```
CHERCHER([Personnes];[Personnes]CodePostal="75@";*) ` Trouver ceux qui habitent Paris…
CHERCHER([Personnes];|;[Personnes]CodePostal="6900@") ` ou Lyon
```

Exemple 12

Recherche par mot-clé : l'exemple suivant recherche dans toute la table [Produits] les enregistrements dont le champ Description contient le mot “facile” :

```
CHERCHER([Produits];[Produits]Description%"facile")
 ` Trouver les produits dont la description contient le mot-clé facile
```

Exemple 13

Nous recherchons les enregistrements correspondant à la réponse fournie dans une boîte de dialogue :

```
vTrouvé:=Demander("Saisissez un code de facture :") `Demander un code de facture à l'utilisateur
Si(OK=1) ` Si l'utilisateur clique sur OK…
   CHERCHER([Factures];[Factures]Code =vTrouvé) `Trouver le code qui correspond à vTrouvé
Fin de si
```

Exemple 14

Cet exemple recherche tous les enregistrements des factures saisies en 1996. Nous recherchons les dates entre le 31/12/95 et le 1/1/97 :

```
CHERCHER([Factures];[Factures]DateFacture >!31/12/95!;*) ` Trouver des factures après le 31/12/95…
CHERCHER([Factures];&;[Factures]DateFacture <!1/1/97!) ` et avant 1/1/97
```

Exemple 15

L'exemple suivant trouve les employés qui ont un salaire entre 20 000 et 40 000 Euros. La recherche inclut les employés qui gagnent 20 000 Euros et exclut ceux qui gagnent 40 000 Euros :

```
CHERCHER([Employés];[Employés]Salaire >=20000;*) ` Trouver les employés qui ont un salaire entre…
CHERCHER([Employés];&;[Employés]Salaire <40000) ` 20 000 et 40 000 Euros
```

Exemple 16

L'exemple suivant cherche les employés du service Marketing qui ont un salaire supérieur à 30 000 Euros. Le champ Salaire est utilisé dans un premier temps car il est indexé. Notez que la seconde recherche utilise un champ venant d'une autre table. Le champ [Service]Nom est lié à la table [Employés] par un lien automatique de N vers 1.

```
CHERCHER([Employés];[Employés]Salaire >30000;*)
 ` Trouver les employés qui ont un salaire supérieur à 30 000 Euros
CHERCHER([Employés];&;[Service]Nom="marketing") ` et qui travaillent dans le service marketing
```

Exemple 17

Soient trois tables reliées par des liens de N vers 1 : [Ville] -> [Département] -> [Région] . La recherche suivante trouve toutes les régions comportant des villes dont le nom débute par "Saint" :

```
CHERCHER([Région];[Ville]Nom="Saint@") ` Trouver toutes les régions contenant des villes commençant par Saint
```

Exemple 18

La recherche suivante recherche les informations égales à la valeur de la variable *mavar*.

```
CHERCHER([Lois];[Lois]Texte =mavar) ` Trouver toutes les lois qui sont égales à la valeur de mavar
```

La recherche peut avoir des résultats différents selon la valeur de *mavar*. Elle sera également exécutée différement. Par exemple :

- Si *mavar* est égale à "Copyright@", la sélection contient toutes les lois qui commencent par Copyright.
- Si *mavar* est égale à "@Copyright@", la sélection contient toutes les lois qui contiennent au moins une occurrence de Copyright.

Exemple 19

L'exemple suivant ajoute ou non les lignes d'une recherche complexe en fonction de la valeur de variables. Ainsi, seuls les critères valides sont pris en compte pour la recherche :

```
CHERCHER([Facture];[Facture]Payee=Faux;*)
Si($ville#"") ` Si un nom de ville a été spécifié
   CHERCHER([Facture];[Facture]Ville_Livraison=$ville;*)
Fin de si
Si($code_postal#"") ` Si un code postal a été spécifié
   CHERCHER([Facture];[Facture]Code_Postal=$code_postal;*)
Fin de si
CHERCHER([Facture]) ` Exécution de la recherche sur les critères
```

Exemple 20

Cet exemple illustre l'utilisation d'un comparateur sous forme d'expression alphanumérique. La valeur du comparateur est définie via un pop up menu placé dans une boîte dialogue de recherche personnalisée :

```
C_TEXTE($ope)
$ope:=_pup_operateur{_pup_operateur} `$ope vaut par exemple "#", ou "="
Si(OK=1)
   CHERCHER([Facture];[Facture]Montant;$ope;$montant)
Fin de si
```

Exemple 21

L'utilisation des index de mots-clés d'image peut accélérer de façon importante vos applications.

```
CHERCHER([IMAGES];[IMAGES]Photos %"cats") // cherche les photos contenant le mot-clé cats
```

Variables et ensembles système

Si la recherche est correctement effectuée, la variable système OK prend la valeur 1.
La variable OK prend la valeur 0 si :

- l'utilisateur clique sur le bouton **Annuler** / **Stop**,
- en mode 'recherche et verrouillage' (cf. commande **FIXER RECHERCHE ET VERROUILLAGE**), la recherche a trouvé au moins un enregistrement verrouillé. Dans ce cas également, l'ensemble système LockedSet est mis à jour.

CHERCHER DANS SELECTION

CHERCHER DANS SELECTION ({laTable }{;}{ critère {; *}})

Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle effectuer la recherche ou ou Table par défaut si ce paramètre est omis
critère	Expression	⇒	Lignes de recherche
*	Opérateur	⇒	Attente d'exécution de la recherche

Description

CHERCHER DANS SELECTION recherche des enregistrements dans *laTable*. **CHERCHER DANS SELECTION** modifie la sélection courante de *laTable* pour le process courant. Le premier enregistrement de la nouvelle sélection devient l'enregistrement courant.

CHERCHER DANS SELECTION a un fonctionnement et des effets proches de ceux de **CHERCHER**. La différence entre ces deux commandes est la portée de la recherche :

- **CHERCHER** recherche des enregistrements dans la table.
- **CHERCHER DANS SELECTION** recherche des enregistrements parmi la sélection courante de la table.

Pour plus d'informations, reportez-vous à la description de la commande **CHERCHER**.

La commande **CHERCHER DANS SELECTION** est utile lorsqu'une recherche ne peut pas être exprimée via une séquence d'appels à **CHERCHER** reliés à l'aide du paramètre *. Typiquement, c'est le cas lorsque vous souhaitez effectuer une recherche dans une sélection courante qui ne résulte pas d'une précédente recherche, mais de l'exécution d'une commande telle que **UTILISER ENSEMBLE**.

Exemple

Vous souhaitez effectuer une recherche parmi les enregistrements préalablement surlignés par l'utilisateur dans un formulaire liste. Vous pouvez écrire :

```
UTILISER ENSEMBLE("UserSet") //remplace la sélection courante par les enregistrements surlignés
CHERCHER DANS SELECTION([Sociétés];[Sociétés]Ville="Paris";*)
CHERCHER DANS SELECTION([Sociétés];[Sociétés]Activité="Affaires boursières")
```

Vous trouvez donc toutes les sociétés basées à Paris, dont l'activité est boursière, parmi la sélection initiale de l'utilisateur.

CHERCHER PAR ATTRIBUT

CHERCHER PAR ATTRIBUT ({laTable}{;}{opConj ;} champObjet ; cheminAttribut ; opRech ; valeur {; *})

Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle la sélection est créée ou Table par défaut si ce paramètre est omis
opConj	Opérateur	⇒	Opérateur à utiliser pour combiner plusieurs requêtes (le cas échéant)
champObjet	Champ	⇒	Champ objet dont les attributs sont à utiliser pour la recherche
cheminAttribut	Chaîne	⇒	Nom ou chemin d'attribut
opRech	Opérateur, Chaîne	⇒	Opérateur de recherche (comparateur)
valeur	Texte, Numérique, Date, Heure	⇒	Valeur à comparer
*	Opérateur	⇒	Attente d'exécution de la recherche

Description

La commande **CHERCHER PAR ATTRIBUT** recherche les enregistrements répondant au(x) critère(s) de recherche spécifié(s) à l'aide des paramètres *champObjet*, *cheminAttribut*, *opRech* et *valeur* et retourne une sélection d'enregistrements de *laTable*.

Note : Pour plus d'informations sur les champs Objet (nouveauté 4D v15), reportez-vous à la section dans le manuel *Mode Développement*.

CHERCHER PAR ATTRIBUT modifie la sélection courante de *laTable* pour le process courant. Le premier enregistrement de la nouvelle sélection devient l'enregistrement courant. Si vous omettez le paramètre *laTable*, la commande s'applique à la table par défaut. Si aucune table par défaut n'a été définie, une erreur est générée.

Le paramètre optionnel *opConj* est utilisé pour combiner plusieurs appels à **CHERCHER PAR ATTRIBUT** en cas de recherche multiple. Les opérateurs de conjonction utilisables sont les mêmes que ceux de la commande **CHERCHER** :

Conjonction	Symbole à utiliser avec CHERCHER PAR ATTRIBUT
ET	&
OU	|
Sauf	#

Le paramètre *opConj* n'est pas nécessaire pour le premier appel à **CHERCHER PAR ATTRIBUT** lors d'une recherche complexe, ou si la recherche ne comporte qu'une ligne. Si vous l'omettez à l'intérieur d'une recherche complexe, le ET (&) est utilisé par défaut.

Dans *champObjet*, passez le champ objet parmi les attributs duquel vous souhaitez effectuer la recherche. Il peut provenir d'une autre table si celle-ci est la table 1 d'une table liée à *laTable* par un lien automatique ou manuel.
CHERCHER PAR ATTRIBUT prend en charge les attributs utilisateurs de 4D Write Pro lorsque les documents sont stockés dans des champs de type Objet. Pour plus d'informations sur ce point, reportez-vous à la section **Stocker les documents 4D Write Pro dans des champs objet 4D**.

Dans *cheminAttribut*, passez le chemin de l'attribut dont vous souhaitez comparer les valeurs pour chaque enregistrement, par exemple "enfants.filles.age". Si vous passez un simple nom, par exemple "lieu", vous désignez l'attribut correspondant situé au premier niveau du champ objet.
Si un attribut "x" est un tableau, **CHERCHER PAR ATTRIBUT** cherchera les enregistrements contenant un attribut "x" dans lequel au moins un élément correspond aux critères. Pour effectuer une recherche parmi les attributs de tableaux, il est nécessaire d'indiquer à la commande **CHERCHER PAR ATTRIBUT** que l'attribut "x" est un tableau en ajoutant "[]" à son nom dans le paramètre *cheminAttribut* (voir exemple 3).

Notes :

- N'oubliez pas que les noms d'attributs tiennent compte des majuscules/minuscules : il est possible d'avoir deux noms d'attributs différents "MonAtt" et "monAtt" dans le même champ d'un enregistrement.
- les noms d'attributs sont "nettoyés" afin d'éliminer les espaces superflus. Par exemple, " mon premier attribut .mon second attribut " est interprété "mon premier attribut .mon second attribut".

Le paramètre *opRech* est l'opérateur qui va permettre de comparer *champObjet* et *valeur*. Vous pouvez utiliser l'un des symboles suivants :

Comparaison	Symbole à utiliser avec CHERCHER PAR ATTRIBUT
Egal à	=
Différent de	#
Inférieur à	<
Supérieur à	>
Inférieur ou égal à	<=
Supérieur ou égal à	>=

Note : Il est possible de définir le comparateur sous la forme d'une expression texte au lieu d'un symbole. Reportez-vous à la description de la commande **CHERCHER** pour plus d'informations.

La *valeur* représente ce qui va être comparé au contenu de *cheminAttribut*. La valeur peut être toute expression du même type que *cheminAttribut*. Le type de la valeur n'est évalué qu'une seule fois, au démarrage de la recherche, et ne l'est donc pas pour chaque enregistrement. Si la recherche porte sur le contenu d'une chaîne de caractères, utilisez dans *valeur* le symbole "@" pour isoler le contenu à rechercher, par exemple "@Dupon@". Il est à noter, dans ce cas, que la recherche ne tire que partiellement parti de l'index (compacité du stockage des données).

Voici la structure type d'une recherche par attribut :

```
CHERCHER PAR ATTRIBUT([Table] ;[Table]ChampObjet ;"attribut1.attribut2";=;valeur)
```

Note : La présence de l'attribut dans le champ objet est un critère implicite pour tous les opérateurs (hormis #). En revanche, pour l'opérateur #, il peut être indéfini (cf. ci-dessous).

Utilisation de l'opérateur # (prise en charge des valeurs Null)

Les recherches par attribut utilisant l'opérateur "#" pourront avoir des résultats différents selon que la propriété est cochée ou non pour le champ objet :

- Propriété **Traduire les NULL en valeurs vides** cochée (option par défaut, recommandée dans la plupart des cas).
  Dans ce cas, l'opérateur "#" doit être perçu comme sélectionnant les enregistrements dont "aucun attribut" du champ ne contient la valeur recherchée.
  Dans ce contexte, 4D considère de façon similaire :
  - les champs pour lesquels la valeur de l'attribut est différente de la valeur recherchée,

- les champs pour lesquels l'attribut n'est pas présent (ou contient la valeur Null).

Par exemple, la recherche suivante retournera les enregistrements des personnes ayant un chien dont le nom n'est pas Médor, ainsi que les enregistrements des personnes n'ayant pas de chien, ou ayant un chien sans nom :

```
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]Animaux;"chien.nom";#;"Médor")
```

Autre exemple : cette recherche retournera tous les enregistrements pour lesquels *[Table]ChampObjet* contient un objet qui contient un attribut *attribut1* qui est lui-même un objet qui contient un attribut *attribut2* dont la valeur n'est pas *valeur*, ainsi que les enregistrements dont le champ objet ne contient pas *attribut1* ou *attribut2* :

```
CHERCHER PAR ATTRIBUT([Table] ;[Table]ChampObjet ;"attribut1.attribut2";#;valeur)
```

Ce principe s'applique également aux attributs tableaux. Par exemple :

```
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]OB_Field;"locations[].city";#;"paris")
```

Cette recherche retournera les enregistrements des personnes n'ayant aucune adresse à Paris.

Pour obtenir spécifiquement les enregistrements dans lesquels l'attribut est indéfini, vous pouvez utiliser un objet vide (cf. exemple 2). A noter toutefois que la recherche de valeurs NULL dans les éléments de tableaux n'est prise en charge.

- Propriété **Traduire les NULL en valeurs vides** non cochée (mode "SQL").
Dans ce cas, les attributs non définis (attributs non présents dans le champ ou ayant la valeur Null) ne sont pas considérés comme équivalents aux valeurs vides par défaut. Par conséquent, les recherches du type "attribut A est différent de valeur B" ne retourneront pas les enregistrements dans lesquels l'attribut A est indéfini.
Pour reprendre l'exemple précédent, lorsque l'option **Traduire les NULL en valeurs vides** n'est pas cochée pour le champ [Personnes]Animaux, la recherche suivante retournera uniquement les enregistrements des personnes ayant un chien dont l'attribut "nom" ne contient pas "Médor". Les enregistrements des personnes n'ayant pas de chien, ou ayant un chien sans nom ne seront pas retournés dans ce cas.

```
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]Animaux;"chien.nom";#;"Médor")
```

Ce fonctionnement, plus proche de la logique SQL, est à réserver pour des besoins spécifiques.

Construire des recherches multiples

Voici les règles à observer pour la construction de recherche par attribut à lignes multiples :

- La première ligne ne doit pas contenir d'opérateur de conjonction,
- Les suivantes peuvent débuter par un opérateur de conjonction. Si vous l'omettez, l'opérateur ET (&) est utilisé par défaut.
- Toutes les lignes, à l'exception de la dernière, doivent s'achever par le symbole *.
- **CHERCHER PAR ATTRIBUT** peut être combiné à des commandes **CHERCHER** classiques (voir exemple).
- Pour lancer la recherche, ne passez pas le paramètre *dans la dernière ligne. Autre solution : vous pouvez exécuter la commande **CHERCHER** sans autre paramètre que la table.

Note : Chaque table maintient sa propre construction de recherche courante. Cela signifie que vous pouvez créer de multiples recherches simultanément, une pour chaque table.

Quelle que soit la manière dont la recherche a été définie :

- Si l'exécution d'une recherche nécessite un certain temps, 4D affiche automatiquement un message contenant un thermomètre de progression. Ce type de message peut être désactivé à l'aide des commandes **LAISSER MESSAGES** et **SUPPRIMER MESSAGES**. Si le thermomètre de progression est affiché, l'utilisateur peut cliquer sur le bouton Stop pour interrompre l'opération. Si la recherche s'est correctement déroulée, la variable système OK prend la valeur 1. Sinon, si la recherche est interrompue, OK prend la valeur 0 (zéro).
- Si des champs objet indexés sont spécifiés, la recherche est optimisée à chaque fois que c'est possible (la recherche commence par les champs indexés), réduisant au maximum la durée de l'opération.

Valeurs date dans l'objet

Les dates sont stockées dans les objets en fonction des paramètres de la base ; par défaut, la *timezone* est prise en compte (voir le sélecteur JSON fuseau horaire local dans la commande **FIXER PARAMETRE BASE**).

```
!1973-05-22! -> "1973-05-21T23:00:00.000Z"
```

Ce paramétrage est également respecté durant les recherches, donc vous n'avez pas à vous en préoccuper si vous utilisez toujours votre base dans la même zone et si les paramètres sont identiques sur chaque machine qui accède aux données. Dans ce contexte, la recherche suivante retournera bien les enregistrements dont l'attribut Anniversaire est égal à !1973-05-22! (stocké "1973-05-21T23:00:00.00Z") :

```
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]OB_Info;"Anniversaire";=;!1973-05-22!)
```

Si vous ne souhaitez pas utiliser le paramétrage GMT, vous pouvez exécuter l'instruction suivante :

```
FIXER PARAMETRE BASE(JSON fuseau horaire local;0)
```

Attention, la portée de ce paramètre est limitée au process. Si vous exécutez cette instruction, le 1er Octobre 1965 sera stocké "1965-10-01T00:00:00.000Z" mais vous devrez fixer le même paramètre avant de lancer vos recherches :

```
FIXER PARAMETRE BASE(JSON fuseau horaire local;0)
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]OB_Info;"Anniversaire";=;1976-11-27!)
```

Utilisation de la propriété virtuelle length

Vous pouvez utiliser la propriété virtuelle "length" avec cette commande. Cette propriété est automatiquement disponible pour tous les attributs de type tableau, et retourne la taille du tableau, c'est-à-dire le nombre d'éléments qu'il contient. Elle peut être utilisée dans le contexte de l'exécution de la commande **CHERCHER PAR ATTRIBUT** (cf. exemple 4).

Exemple 1

Dans cet exemple, l'attribut "age" est soit une chaîne soit un entier et nous souhaitons trouver les personnes dont l'âge est situé entre 20 et 29. Les deux premières lignes interrogent l'attribut en tant qu'entier (>=20 et < 30) et les suivantes interrogent l'attribut en tant que chaîne (débute par "2" mais est différent de "2").

```
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]OB_Info;"age";>=;20;*)
CHERCHER PAR ATTRIBUT([Personnes]; & ;[Personnes]OB_Info;"age";<;30;*)
CHERCHER PAR ATTRIBUT([Personnes];|;[Personnes]OB_Info;"age";=;"2@";*)
CHERCHER PAR ATTRIBUT([Personnes]; & ;[Personnes]OB_Info;"age";#;"2") //pas de * final pour lancer l'exécution
```

Exemple 2

La commande **CHERCHER PAR ATTRIBUT** peut être utilisée pour rechercher des enregistrements dans lesquels certains attributs sont définis (ou non définis). Pour cela, vous devez utiliser un objet vide :

```
 //Trouver les enregistrements où l'email est défini dans le champ objet
C_OBJET($undefined)
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]Info;"email";#;$undefined)
```

```
 //Trouver les enregistrements où le zip code n'est PAS défini dans le champ objet
C_OBJET($undefined)
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]Info;"zip code";=;$undefined)
```

Note : Cette syntaxe spécifique n'est pas prise en charge avec les attributs de type tableau. La recherche de valeurs NULL dans les attributs de tableau donne des résultats invalides.

Exemple 3

Vous voulez chercher un champ contenant des attributs tableaux. Avec les deux enregistrements suivants :

Enregistrement 1 :
[Personnes]nom : "martin"
[Personnes]OBFied :
```
  "locations" : [ {
        "kind":"office",
        "city":"paris"
      } ]
```
Enregistrement 2 :
[Personnes]nom : "smith"
[Personnes]OBFied :
```
  "locations" : [ {
        "kind":"home",
        "city":"lyon"
      } , {
        "kind":"office",
        "city":"paris"
      } ]
```
... **CHERCHER PAR ATTRIBUT** trouvera les personnes ayant une localisation à "paris" par cette recherche :

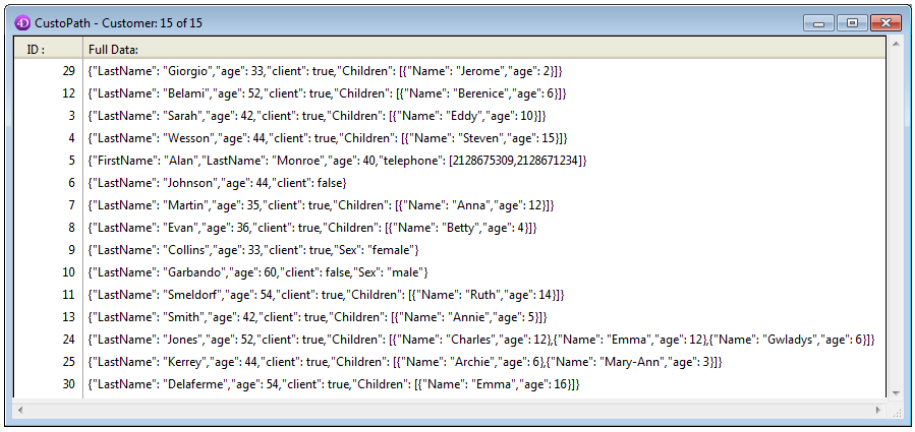
```
 //on indique l'attribut tableau avec la syntaxe "[]"
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]OB_Field;"locations[].city";=;"paris")
 //trouve "martin" et "smith"
```

Note : Si vous avez défini plusieurs critères sur le même attribut tableau, les critères correspondants ne s'appliqueront pas nécessairement au même élément de tableau. Dans l'exemple ci-dessous, la recherche retournera "smith" car l'attribut a un élément "locations" dont le "kind" est "home" et un élément "locations" dont le "city" est "paris", même s'il ne s'agit pas du même élément :

```
CHERCHER PAR ATTRIBUT([Personnes];[Personnes]OB_Field;"locations[].kind";=;"home";*)
CHERCHER PAR ATTRIBUT([Personnes]; & ;[Personnes]OB_Field;"locations[].city";=;"paris")
 //trouve "smith"
```

Exemple 4

Cet exemple illustre l'utilisation de la propriété virtuelle "length". Votre base de données comporte un champ objet [Customer]full_Data avec les données suivantes :



Vous souhaitez obtenir les enregistrements des clients qui ont deux enfants ou plus. Vous pouvez écrire :

```
CHERCHER PAR ATTRIBUT([Customer];[Customer]full_Data;"Children.length";>=;2)
```

Variables et ensembles système

Si la recherche est correctement effectuée, la variable système OK prend la valeur 1.
La variable OK prend la valeur 0 si :

- l'utilisateur clique sur le bouton **Annuler** / **Stop**,
- en mode 'recherche et verrouillage' (cf. commande **FIXER RECHERCHE ET VERROUILLAGE**), la recherche a trouvé au moins un enregistrement verrouillé. Dans ce cas également, l'ensemble système LockedSet est mis à jour.

CHERCHER PAR ATTRIBUT DANS SELECTION

CHERCHER PAR ATTRIBUT DANS SELECTION ({laTable}{;}{opConj ;} champObjet ; cheminAttribut ; opRecherche ; valeur {; *})

Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle la sélection est créée ou Table par défaut si ce paramètre est omis
opConj	Opérateur	⇒	Opérateur à utiliser pour combiner plusieurs requêtes (le cas échéant)
champObjet	Champ	⇒	Champ objet dont les attributs sont à utiliser pour la recherche
cheminAttribut	Chaîne	⇒	Nom ou chemin d'attribut
opRecherche	Opérateur, Chaîne	⇒	Opérateur de recherche (comparateur)
valeur	Texte, Numérique, Date, Heure	⇒	Valeur à comparer
*	Opérateur	⇒	Attente d'exécution de la recherche

Description

CHERCHER PAR ATTRIBUT DANS SELECTION fonctionne de la même façon et exécute les mêmes actions que la commande **CHERCHER PAR ATTRIBUT**. La différence entre ces deux commandes est la portée de la recherche :

- **CHERCHER PAR ATTRIBUT** recherche les enregistrements sur la totalité des enregistrements de la table.
- **CHERCHER PAR ATTRIBUT DANS SELECTION** recherche les enregistrements dans la sélection courante de la table.

CHERCHER PAR ATTRIBUT DANS SELECTION recherche des enregistrements dans *laTable*. La commande **CHERCHER PAR ATTRIBUT DANS SELECTION** change la sélection courante de *laTable* pour le process courant et le premier enregistrement de la sélection devient l'enregistrement courant.

Pour plus d'information, voir la description de la commande **CHERCHER PAR ATTRIBUT**.

La commande **CHERCHER PAR ATTRIBUT DANS SELECTION** est utile lorsqu'une recherche ne peut pas être définie en utilisant la combinaison de plusieurs **CHERCHER PAR ATTRIBUT** (voire de plusieurs **CHERCHER**) appelées conjointement avec le paramètre *. C'est typiquement le cas lorsque vous recherchez dans une sélection courante qui ne résulte pas d'une recherche mais d'une commande telle que **UTILISER ENSEMBLE**.

Exemple

Vous souhaitez trouver les personnes âgées entre 20 et 30 ans parmi les enregistrements sélectionnés par l'utilisateur :

```
UTILISER ENSEMBLE("UserSet") // crée une nouvelle sélection courante
CHERCHER PAR ATTRIBUT DANS SELECTION([People];[People]OB_Info;"age";>;20;*)
CHERCHER PAR ATTRIBUT DANS SELECTION([People];&;[People]OB_Info;"age";<;30) //déclenche la recherche
```

CHERCHER PAR EXEMPLE

CHERCHER PAR EXEMPLE ({laTable}{;}{*})

Paramètre	Type		Description
laTable	Table	➡	Table de laquelle une sélection d'enregistrements doit être retournée ou Table par défaut si ce paramètre est omis
*	Opérateur	➡	Masquer les barres de défilement

Description

La commande **CHERCHER PAR EXEMPLE** effectue la même action que la commande de menu **Recherche par formulaire**... en mode Développement. Cette commande affiche le formulaire entrée courant comme fenêtre de recherche. **CHERCHER PAR EXEMPLE** cherche dans *laTable* les données que l'utilisateur a saisies dans cette fenêtre. Le formulaire doit contenir les champs sur lesquels vous voulez que l'utilisateur puisse effectuer la recherche. La recherche est optimisée : les champs indexés sont automatiquement utilisés.
Si vous passez le paramètre optionnel *, les barres de défilement du formulaire sont masquées.

Reportez-vous au manuel Mode Développement de 4D pour plus d'informations sur l'utilisation de la commande de menu **Recherche par formulaire...** du mode Développement.

Exemple

La méthode dans l'exemple suivant affiche le formulaire *maRecherche*. Si l'utilisateur valide le formulaire et exécute la recherche (c'est-à-dire si la variable système OK prend la valeur 1), les enregistrements trouvés sont affichés :

```
FORM FIXER ENTREE([Personnes];"maRecherche") ` Ce formulaire devient le formulaire entrée
CHERCHER PAR EXEMPLE([Personnes]) ` Afficher le formulaire pour la recherche
Si(OK=1) ` Si l'utilisateur valide la recherche
   VISUALISER SELECTION([Personnes]) ` Visualiser les enregistrements trouvés
Fin de si
```

Variables et ensembles système

Si l'utilisateur clique sur le bouton Valider ou appuie sur la touche Entrée, la variable système OK prend la valeur 1 et la recherche est effectuée. Si l'utilisateur clique sur le bouton Annuler ou utilise la touche d'annulation, la variable système OK prend la valeur 0 et la recherche est annulée.

CHERCHER PAR FORMULE

CHERCHER PAR FORMULE (laTable {; formule})

Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle effectuer la recherche
formule	Booléen	⇒	Formule de recherche

Description

CHERCHER PAR FORMULE effectue une recherche d'enregistrements dans *laTable*. **CHERCHER PAR FORMULE** modifie la sélection courante de *laTable* pour le process courant et fait du premier enregistrement le nouvel enregistrement courant.

CHERCHER PAR FORMULE et la commande **CHERCHER PAR FORMULE DANS SELECTION** fonctionnent exactement de la même manière, à la différence près que **CHERCHER PAR FORMULE** effectue sa recherche parmi la totalité des enregistrements de la table alors que **CHERCHER PAR FORMULE DANS SELECTION** se cantonne aux enregistrements de la sélection courante.

Les deux commandes appliquent *formule* à chaque enregistrement de la table ou de la sélection. *formule* est une expression booléenne qui doit retourner **VRAI** ou **FAUX**. Si *formule* retourne **Vrai**, l'enregistrement est inclus dans la nouvelle sélection.

formule peut être simple (par exemple la comparaison d'un champ à une valeur) ou complexe (réalisation d'un calcul ou même évaluation de valeurs dans une table liée). Ce peut être une fonction 4D, ou une fonction ou une expression que vous avez créée. Lorsque vous travaillez avec des champs de type Alpha ou Texte, vous pouvez utiliser dans *formule* des jokers (@) ainsi que l'opérateur "contient" (%) pour la recherche par mots-clés. Pour plus d'informations, reportez-vous à la description de la commande **CHERCHER**.

Si vous omettez le paramètre *formule*, 4D affiche la boîte de dialogue de recherche (l'utilisateur peut ajouter une ligne de formule en effectuant **Alt+clic** sur le bouton **[+]**).

Lorsque la recherche est terminée, le premier enregistrement de la nouvelle sélection est chargé depuis le disque et devient l'enregistrement courant.

Ces commandes sont optimisées et peuvent notamment tirer parti des index. Lorsque le type de requête le permet, ces commandes exécutent des requêtes équivalentes à un **CHERCHER**. Par exemple, l'instruction **CHERCHER PAR FORMULE([matable]; [matable]monchamp=valeur)** sera exécutée comme **CHERCHER([matable]; [matable]monchamp=valeur)**, ce qui permettra d'utiliser l'index. 4D pourra également optimiser les requêtes contenant des parties non optimisables, en exécutant d'abord les parties optimisables puis en combinant les résultats avec le reste de la requête. Par exemple, l'instruction **CHERCHER PAR FORMULE([matable];Longueur(monchamp)=valeur)** ne sera pas optimisée. En revanche, **CHERCHER PAR FORMULE([matable];Longueur(monchamp)=valeur1 | monchamp=valeur2)** sera partiellement optimisée.

Ces commandes effectuent par défaut des "jointures" à la manière du SQL lorsque vous comparez des champs de tables différentes. Avec ce principe, les recherches sont optimisées et il n'est pas nécessaire qu'un lien automatique structurel existe entre les tables. Par exemple, vous pouvez exécuter une instruction du type **CHERCHER PAR FORMULE([Table_A];([Table_A]champ_X = [Table_B]champ_Y) & ([Table_B]champ_Y = "abc"))** (cf. exemple 3). La première partie de la formule (*[Table_A]champ_X = [Table_B]champ_Y*) établit la jointure entre les deux champs et la seconde partie (*[Table_B]champ_Y = "abc"*) définit le ou les critère(s) de recherche (au moins un critère doit être défini).
S'ils existent, les liens entre les tables ne sont en principe pas utilisés. Toutefois, ces commandes utilisent les liens automatiques dans les cas suivants :
- si la formule ne peut se décomposer en éléments de la forme *{ champ ; comparateur ; valeur}*
- si deux champs de la même table sont comparés.

Note : Pour des raisons de compatibilité, il est possible de désactiver le mécanisme de jointures, soit globalement via les Propriétés de la base (bases de données converties uniquement) soit par process via la commande **FIXER PARAMETRE BASE**.

4D Server : A compter de la version 11 de 4D Server, ces commandes sont exécutées sur le serveur, ce qui optimise leur exécution. A noter qu'en cas d'appel direct de variables dans la *formule*, la requête est calculée avec la valeur de la variable sur le poste client. Par exemple, l'instruction **CHERCHER PAR FORMULE([matable];[matable]monchamp=mavariable)** sera exécutée sur le serveur mais avec le contenu de la variable *mavariable* du client. En revanche, ce principe n'est pas appliqué pour les formules utilisant des méthodes qui, elles-mêmes, font appel à des variables (la valeur des variables est évaluée sur le serveur). Dans ce contexte, il peut être judicieux d'utiliser l'attribut de méthode "Exécuter sur serveur" permettant d'exécuter une méthode sur le serveur en lui passant des paramètres (variables) (cf. manuel Mode Développement).
Dans les versions précédentes de 4D Server, ces commandes étaient exécutées sur les postes clients. Par compatibilité, ce fonctionnement est maintenu dans les bases de données converties en version 11. Une préférence de compatibilité et un sélecteur de la commande **FIXER PARAMETRE BASE** permettent toutefois d'adopter le fonctionnement de la version 11 (exécution sur le serveur) dans les bases de données converties.

Exemple 1

L'exemple suivant recherche les enregistrements de toutes les factures qui ont été saisies au mois de décembre, sans tenir compte de l'année. Le principe est d'appliquer la fonction **Mois de** à chaque enregistrement. Cette recherche ne pourrait pas être effectuée d'une autre manière sans créer un champ séparé pour le mois :

```
CHERCHER PAR FORMULE([Factures];Mois de([Factures]Saisie)=12)
  ` Chercher les factures saisies en décembre
```

Exemple 2

L'exemple suivant recherche les enregistrements de toutes les personnes dont le nom comporte plus de dix caractères :

```
CHERCHER PAR FORMULE([Personnes];Longueur([Personnes]Nom)>10)
  ` Chercher les personnes dont le nom fait plus de dix caractères
```

Exemple 3

Cet exemple active les jointures SQL pour une recherche par formule spécifique :

```
$valcourante:=Lire parametre base(Jointures CHERCHER PAR FORMULE)
```

```
FIXER PARAMETRE BASE(Jointures CHERCHER PAR FORMULE;2) //Activer les jointures SQL
 //Chercher toutes les lignes de factures du client "ACME" alors que les tables ne sont pas liées
CHERCHER PAR FORMULE([ligne_factures];([ligne_factures]facture_id=[facture]id) & ([facture]client="ACME"))
FIXER PARAMETRE BASE(Jointures CHERCHER PAR FORMULE;$valcourante) //on rétablit le paramétrage courant
```

CHERCHER PAR FORMULE DANS SELECTION

CHERCHER PAR FORMULE DANS SELECTION (laTable {; formule})

Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle effectuer la recherche parmi la sélection courante
formule	Booléen	⇒	Formule de recherche

Description

La commande **CHERCHER PAR FORMULE DANS SELECTION** vous permet de rechercher des enregistrements dans *laTable*. **CHERCHER PAR FORMULE DANS SELECTION** modifie la sélection courante de *laTable* pour le process courant et fait du premier enregistrement le nouvel enregistrement courant.
CHERCHER PAR FORMULE DANS SELECTION fonctionne de la même manière que **CHERCHER PAR FORMULE**. La différence entre ces deux commandes se situe au niveau de la portée de la recherche :

- **CHERCHER PAR FORMULE** effectue sa recherche parmi la totalité des enregistrements de la table.
- **CHERCHER PAR FORMULE DANS SELECTION** effectue sa recherche uniquement parmi les enregistrements de la sélection courante.

Pour plus d'informations, reportez-vous à la description de la commande **CHERCHER PAR FORMULE**.

CHERCHER PAR TABLEAU

CHERCHER PAR TABLEAU (champCible ; tableau)

Paramètre	Type		Description
champCible	Champ	⇒	Champ duquel comparer les valeurs
tableau	Tableau	⇒	Tableau des valeurs recherchées

Description

La commande **CHERCHER PAR TABLEAU** recherche dans la table du champ passé en premier paramètre tous les enregistrements pour lesquels la valeur de *champCible* est égale à au moins une des valeurs des éléments du tableau *tableau*. Les enregistrements trouvés constituent la nouvelle sélection courante.

Cette commande permet de construire rapidement et simplement une recherche sur plusieurs valeurs.

Notes :

- Cette commande ne peut pas être utilisée avec des champs de type image et BLOB.
- *champCible* et *tableau* doivent impérativement être du même type. Exception : vous pouvez utiliser un tableau de type Entier long avec un champ de type Heure.

Exemple

Cet exemple permet de récupérer les enregistrements des clients français et américains :

```
TABLEAU ALPHA(2;TabRecherche;2)
TabRecherche{1}:="FR"
TabRecherche{2}:="US"
CHERCHER PAR TABLEAU([Clients]Pays;TabRecherche)
```

CHERCHER PAR TABLEAU DANS SELECTION

CHERCHER PAR TABLEAU DANS SELECTION (champCible ; tableau)

Paramètre	Type		Description
champCible	Champ	⇒	Champ duquel comparer les valeurs
tableau	Tableau	⇒	Tableau des valeurs recherchées

Description

La commande **CHERCHER PAR TABLEAU DANS SELECTION** recherche dans la sélection courante de la table du champ passé en premier paramètre les enregistrements pour lesquels la valeur de *champCible* est égale à au moins une des valeurs des éléments du *tableau*. Les enregistrements trouvés constituent la nouvelle sélection courante.

CHERCHER PAR TABLEAU DANS SELECTION fonctionne de la même manière que **CHERCHER PAR TABLEAU**. La différence entre ces deux commandes se situe au niveau de la portée de la recherche :

- **CHERCHER PAR TABLEAU** effectue sa recherche parmi la totalité des enregistrements de la table de *champCible*.
- **CHERCHER PAR TABLEAU DANS SELECTION** effectue sa recherche uniquement parmi les enregistrements de la sélection courante de la table de *champCible*.

Pour plus d'informations, reportez-vous à la description de la commande **CHERCHER PAR TABLEAU**.

DECRIRE EXECUTION RECHERCHE

DECRIRE EXECUTION RECHERCHE (statut)

Paramètre	Type		Description
statut	Booléen	⇨	Vrai=Enregistrer la description des requêtes, Faux=Stopper l'enregistrement

Description

La commande **DECRIRE EXECUTION RECHERCHE** permet d'activer ou d'inactiver le mode d'analyse de l'exécution des recherches pour le process courant. La commande fonctionne uniquement dans le contexte des commandes de recherche du langage 4D telles que **CHERCHER**.

L'appel de la commande avec le paramètre *statut* à **Vrai** active le mode d'analyse des recherches. Dans ce mode, le moteur de 4D enregistrera en interne deux séries d'informations spécifiques lors de chaque requête effectuée par la suite sur les données :

- la description détaillée de la recherche juste avant son exécution, c'est-à-dire la recherche prévue (le plan de recherche),
- la description détaillée de la recherche telle qu'elle a réellement été exécutée (le chemin de recherche).

Les informations enregistrées incluent le type de recherche (indexée, séquentielle), le nombre d'enregistrements trouvés et le temps nécessaire à l'exécution de chaque critère de recherche. Vous pouvez ensuite lire ces informations à l'aide des commandes **Lire dernier plan recherche** et **Lire dernier chemin recherche**.

En général, la description du plan d'une recherche et celle de son chemin sont identiques, mais elles peuvent toutefois différer car 4D peut mettre en oeuvre des optimisations dynamiques au cours de l'exécution de la recherche, dans le but d'améliorer les performances. Par exemple, une recherche indexée peut être convertie dynamiquement en recherche séquentielle si le moteur de 4D estime qu'elle sera plus rapide — c'est le cas notamment lorsque le nombre d'enregistrements parmi lesquels effectuer la recherche est faible.

Passez **Faux** dans le paramètre *statut* lorsque vous n'avez plus besoin d'analyser les recherches. Le mode d'analyse de l'exécution des recherches peut ralentir l'application.

Exemple

L'exemple suivant illustre le type d'information obtenue via ces commandes :

```
C_TEXTE($vResultPlan;$vResultPath)
DECRIRE EXECUTION RECHERCHE(Vrai)  //mode analyse
CHERCHER([Employés];[Employés]Nom="T@";*)  // Chercher les employés dont le nom débute par T...
CHERCHER([Employés]; & ;[Sociétés]Nom="H@";*) // travaillant pour une société dont le nom débute par H
CHERCHER([Employés]; & ;[Employés]Salaire>2500;*) // dont le salaire est > 2500
CHERCHER([Employés]; & ;[Villes]nbHab<50000)  // habitant dans une ville de moins de 50000 habitants
$vResultPlan:=Lire dernier plan recherche(Description format Texte)
$vResultPath:=Lire dernier chemin recherche(Description format Texte)
DECRIRE EXECUTION RECHERCHE(Faux)  //Fin du mode analyse
```

A l'issue de l'exécution de ce code, *$vResultPlan* et *$vResultPath* contiennent les descriptions des recherches effectuées, par exemple :

```
$vResultPlan :
   Employés.Nom == T@ And Employés.Salaire > 2500 And Join on Table : Sociétés  :  Employés.Société =
Sociétés.Nom [index : Sociétés.Nom ] LIKE H@ And Join on Table : Villes  :  Employés.Ville = Villes.Nom [index :
Villes.nbHab ] < 50000
$vResultPath :
(Employés.Nom == T@ And Employés.Salaire > 2500) And (Join on Table : Sociétés  :  Employés.Société =
Sociétés.Nom with filter {[index : Sociétés.Nom ] LIKE H@}) And (Join on Table : Villes  :  Employés.Ville =
Villes.Nom with filter {[index : Villes.nbHab ] < 50000})   (3 records found in 1 ms)
```

Si la constante Description format XML est passée à la commande **Lire dernier chemin recherche**, *$vResultPath* contient la description de la recherche exprimée en XML :

```
$vResultPath :    <QueryExecution>       <steps description="And" time="0" recordsfounds="1227">          <steps
description="[Merge] : ACTORS with CITIES" time="13" recordsfounds="1227">            <steps description="[Join]
: ACTORS.Birth_City_ID =CITIES.City_ID" time="13" recordsfounds="1227"/>          </steps>       </steps>
   </QueryExecution>
```

FIXER DESTINATION RECHERCHE

FIXER DESTINATION RECHERCHE (destinationType {; destinationObjet {; destinationPtr}})

Paramètre	Type		Description
destinationType	Entier long	⇒	0=sélection courante, 1=ensemble, 2=sélection temporaire, 3=variable
destinationObjet	Chaîne, Variable	⇒	Nom de l'ensemble ou Nom de la sélection temporaire ou Variable
destinationPtr	Pointeur	⇒	Pointeur vers la variable locale si destinationType=3

Description

La commande **FIXER DESTINATION RECHERCHE** vous permet d'indiquer à 4D où placer les résultats de toutes les recherches qui suivent l'appel de cette commande dans le process courant.

Vous spécifiez le type de la destination dans le paramètre *destinationType*. 4D fournit les constantes prédéfinies suivantes, placées dans le thème **Recherches** :

Constante	Type	Valeur
Vers ensemble	Entier long	1
Vers sélection courante	Entier long	0
Vers sélection temporaire	Entier long	2
Vers variable	Entier long	3

Vous spécifiez le nom de la destination de la recherche dans le paramètre optionnel *destinationObjet* en fonction du tableau suivant :

Paramètre destinationType	Paramètre destinationObjet
0 (sélection courante)	Vous ne passez pas de paramètre.
1 (ensemble)	Vous passez le nom de l'ensemble (existant ou à créer)
2 (sélection temporaire)	Vous passez le nom de la sélection temporaire (existante ou à créer)
3 (variable)	Vous passez soit une variable numérique (existante ou à créer), soit une chaîne vide "" pour utiliser le paramètre *destinationPtr*

Avec

```
FIXER DESTINATION RECHERCHE(Vers sélection courante)
```

Les enregistrements trouvés par la recherche seront placés dans la sélection courante de la table dans laquelle la recherche est effectuée.

Avec

```
FIXER DESTINATION RECHERCHE(Vers ensemble;"monEnsem")
```

Les enregistrements trouvés par la recherche seront placés dans l'ensemble *monEnsem*. La sélection courante et l'enregistrement courant de la table dans laquelle vous recherchez restent inchangés.

Avec

```
FIXER DESTINATION RECHERCHE(Vers sélection temporaire;"maTemp")
```

Les enregistrements trouvés par la recherche seront placés dans la sélection temporaire *maTemp*. La sélection courante et l'enregistrement courant pour la table sur laquelle vous effectuez la recherche restent inchangés.

Notes :

- Si la sélection temporaire n'existe pas avant l'appel, elle est automatiquement créée une fois la recherche effectuée.
- Cette commande gère les sélections temporaires comme la commande [#cmd id="334"/] : seules des références sont conservées. Une fois la sélection temporaire utilisée, elle n'existe plus.

Avec

```
FIXER DESTINATION RECHERCHE(Vers variable;$vlRésultatRech)
```

Ou

```
FIXER DESTINATION RECHERCHE(Vers variable;"";->$vlRésultatRech)
```

Note : Cette seconde syntaxe facilite l'utilisation conjointe de la commande avec **LIRE DESTINATION RECHERCHE**.

Le **nombre** d'enregistrements trouvés par la recherche sera placé dans la variable *$vlRésultatRech*. La sélection courante et l'enregistrement courant de la table dans laquelle vous effectuez la recherche restent inchangés.

Attention : FIXER DESTINATION RECHERCHE affecte toutes les recherches suivantes dans le process courant. N'oubliez pas d'associer toujours un appel à **FIXER DESTINATION RECHERCHE** (lorsque *destinationType#0*) à un appel à **FIXER DESTINATION RECHERCHE(0)** ultérieur pour rétablir le mode standard de recherche.

FIXER DESTINATION RECHERCHE modifie uniquement le comportement des commandes de recherche, c'est-à-dire :

- **CHERCHER**
- **CHERCHER DANS SELECTION**
- **CHERCHER PAR EXEMPLE**
- **CHERCHER PAR FORMULE**
- **CHERCHER PAR FORMULE DANS SELECTION**

- CHERCHER PAR SQL
- CHERCHER PAR TABLEAU
- CHERCHER PAR TABLEAU DANS SELECTION
- CHERCHER PAR ATTRIBUT
- CHERCHER PAR ATTRIBUT DANS SELECTION

En revanche, **FIXER DESTINATION RECHERCHE** n'affecte pas les autres commandes qui modifient la sélection courante telles que **TOUT SELECTIONNER**, **LIEN RETOUR** , etc.

Exemple 1

Vous créez un formulaire qui affiche les enregistrements de la table *[Annuaire]*. Vous créez un objet de type onglet nommé *asRolodex* (avec un onglet pour chaque lettre de l'alphabet) et un sous-formulaire qui affiche les enregistrements de la table *[Annuaire]*. En choisissant un onglet, vous affichez les enregistrements qui correspondent à cette lettre. Puisque, dans cet exemple, la table *[Annuaire]* contient des données statiques, vous ne voulez pas effectuer une recherche chaque fois que vous cliquez sur un onglet et donc vous dépensez moins de temps précieux à exécuter ces recherches. Pour faire ceci, vous pouvez placer vos recherches dans les sélections temporaires pour les réutiliser quand il le faut. Vous écrivez la méthode objet de l'onglet *asRolodex* comme indiquée ci-dessous :

```
   ` Méthode objet de l'onglet asRolodex
Au cas ou

   :(Evenement formulaire=Sur chargement)
   ` Avant que le formulaire s'affiche à l'écran,
   ` initialiser l'onglet et le tableau de booléens qui nous indiquent
   ` si une recherche pour la lettre sur laquelle vous avez cliqué
   ` a été exécutée ou pas
      TABLEAU ALPHA(1;asRolodex;26)
      TABLEAU BOOLEEN(abRechFini;26)
      Boucle($vlElém;1;26)
         asRolodex{$vlElém}:=Caractere(64+$vlElém)
         abRechFini{$vlElém}:=Faux
      Fin de boucle

   :(Evenement formulaire=Sur clic)
   ` Lorsque l'utilisateur clique sur un onglet, vérifier si une recherche pour cette lettre
   ` a été exécutée ou pas
      Si(Non(abRechFini{asRolodex}))
   ` Sinon, fixer la destination de la recherche vers une sélection temporaire
         FIXER DESTINATION RECHERCHE(Vers sélection temporaire;"temp")
   ` Effectuer la recherche
         CHERCHER([Phone Book];[Phone Book]Last name=asRolodex{asRolodex}+"@")
   ` Restituer le mode standard de recherche
         FIXER DESTINATION RECHERCHE(Vers sélection courante)
   ` Utiliser les enregistrements trouvés
         UTILISER SELECTION("temp")
         COPIER SELECTION([Phone Book];"Rolodex"+asRolodex{asRolodex})
   ` La prochaine fois que cette lettre est choisie, la recherche ne sera pas exécutée
         abRechFini{asRolodex}:=Vrai
      Sinon
   ` Utiliser la sélection temporaire existante pour l'affichage des enregistrements qui correspondent à cette
lettre
         UTILISER SELECTION("Rolodex"+asRolodex{asRolodex})
      Fin de si

   :(Evenement formulaire=Sur libération)
   ` Après que le formulaire disparaît de l 'écran
   ` Effacer les sélections temporaires de la mémoire
      Boucle($vlElem;1;26)
         Si(abRechFini{$vlElém})
            EFFACER SELECTION("Rolodex"+asRolodex{$vlElém})
         Fin de si
      Fin de boucle
   ` Effacer les deux tableaux dont nous n'avons pas besoin
      EFFACER VARIABLE(asRolodex)
      EFFACER VARIABLE(abRechFini)

Fin de cas
```

Exemple 2

La méthode ***ValeursUniques*** suivante vérifie si les valeurs sont uniques pour des champs dans une table de votre choix. L'enregistrement courant peut déjà exister ou vient d'être créé.

```
   ` Méthode projet ValeursUniques
   ` ValeursUniques ( Pointeur ; Pointeur { ; Pointeur... } ) -> Booléen
   ` ValeursUniques ( ->Table ; ->Champ { ; ->Champ2... } ) -> Oui ou non

C_BOOLEEN($0)
```

```
C_POINTEUR($1)
C_ENTIER LONG($vlChamp;$vlNmbChamps;$vlTrouvé;$vlEnregCour)
$vlNmbChamps:=Nombre de paramètres-1
$vlEnregCour:=Numero enregistrement($1->)
Si($vlNmbChamps>0)
   Si($vlEnregCour#-1)
      Si($vlEnregCour<0)
 ` Il s'agit d'un nouvel enregistrement qui n'a pas été sauvegardé (numéro d'enregistrement est
 ` égal à -3)
 ` donc nous pouvons arrêter la recherche dès que nous avons trouvé un enregistrement
         FIXER LIMITE RECHERCHE(1)
      Sinon
 ` Il s'agit d'un enregistrement existant, donc nous pouvons arrêter
 ` la recherche dès que nous avons trouvé au moins deux enregistrements
         FIXER LIMITE RECHERCHE(2)
      Fin de si
 ` La recherche retournera le résultat dans la variable $vlTrouvé
 ` sans changer l'enregistrement courant ni la sélection courante
      FIXER DESTINATION RECHERCHE(Vers variable;$vlTrouvé)
 ` Construire la recherche selon le nombre de champs spécifiés
      Au cas ou
         :($vlNmbChamps=1)
            CHERCHER($1->;$2->=$2->)
         :($vlNmbChamps=2)
            CHERCHER($1->;$2->=$2->;*)
            CHERCHER($1->; & ;$3->=$3->)
         Sinon
            CHERCHER($1->;$2->=$2->;*)
            Boucle($vlChamp;2;$vlNmbChamps-1)
               CHERCHER($1->; & ;${1+$vlChamp}->=${1+$vlChamp}->;*)
            Fin de boucle
            CHERCHER($1->; & ;${1+$vlNmbChamps}->=${1+$vlNmbChamps}->)
      Fin de cas
      FIXER DESTINATION RECHERCHE(0) ` Rétablir le mode standard de recherche
      FIXER LIMITE RECHERCHE(0) ` Enlever la limite sur la recherche
 ` Traiter le résultat de la recherche
      Au cas ou
         :($vlTrouvé=0)
            $0:=Vrai  ` Pas de valeurs dupliquées
         :($vlTrouvé=1)
            Si($vlEnregCour<0)
               $0:=Faux ` Trouvé un enregistrement existant avec les mêmes valeurs que le nouveau
            Sinon
               $0:=Vrai  ` Pas de valeurs dupliquées, nous avons trouvé le même enregistrement
            Fin de si
         :($vlTrouvé=2)
            $0:=Faux   ` Quoi que ce soit, les valeurs sont dupliquées
      Fin de cas
   Sinon
      Si(◊Débogage)   ` Cela n'a aucun sens, signalez-le pendant le développement de la base
         TRACE   ` ATTENTION ! Cette méthode a été appelée sans enregistrement courant
      Fin de si
      $0:=Faux   ` Ne peut pas garantir le résultat
   Fin de si
Sinon
   Si(◊Débogage)   ` Cela n'a aucun sens, signalez-le pendant le développement de la base
      TRACE   ` ATTENTION ! Cette méthode a été appelée sans conditions de recherche
   Fin de si
   $0:=Faux   ` Ne peut pas garantir le résultat
Fin de si
```

Lorsque cette méthode est implémentée dans votre application, vous pouvez écrire le code suivant :

```
 ` ...
Si(ValeursUniques(->[Contacts];->[Contacts]Société;->[Contacts]Nom;->[Contacts]Prénom))
 ` Traitement de l'enregistrement qui a les valeurs uniques
Sinon
   ALERTE("Il existe déjà un contact avec ce nom pour cette société.")
Fin de si
 ` ...
```

FIXER LIMITE RECHERCHE

FIXER LIMITE RECHERCHE (limite)

Paramètre	Type		Description
limite	Entier long	⇨	Nombre limite d'enregistrements ou 0 pour nombre illimité

Description

La commande **FIXER LIMITE RECHERCHE** vous permet d'indiquer à 4D d'arrêter toutes les recherches suivant l'appel de cette commande dans le process courant dès que le nombre d'enregistrements défini dans *limite* a été atteint.

Si, par exemple, *limite* est égal à 1, les recherches s'arrêteront dès qu'un enregistrement sera trouvé selon les conditions de la recherche.

Pour que les recherches soient de nouveau sans limite, appelez **FIXER LIMITE RECHERCHE** en fixant le paramètre *limite* à 0.

Attention : FIXER LIMITE RECHERCHE affecte toutes les recherches dans le process courant. N'oubliez pas d'associer toujours un appel à **FIXER LIMITE RECHERCHE(limite)** (lorsque *limite>0*) à un appel à **FIXER LIMITE RECHERCHE(0)** ultérieur pour rétablir les recherches sans limite.

FIXER LIMITE RECHERCHE modifie uniquement le comportement des commandes de recherche, c'est-à-dire :

- **CHERCHER**
- **CHERCHER DANS SELECTION**
- **CHERCHER PAR EXEMPLE**
- **CHERCHER PAR FORMULE**
- **CHERCHER PAR FORMULE DANS SELECTION**
- **CHERCHER PAR SQL**
- **CHERCHER PAR TABLEAU**
- **CHERCHER PAR TABLEAU DANS SELECTION**
- **CHERCHER PAR ATTRIBUT**
- **CHERCHER PAR ATTRIBUT DANS SELECTION**

En revanche, **FIXER LIMITE RECHERCHE** n'affecte pas les autres commandes qui modifient la sélection courante d'une table telles que **TOUT SELECTIONNER**, **LIEN RETOUR**, etc.

Exemple 1

Pour effectuer une recherche qui correspond à la formule "...trouver dix clients avec lesquels les ventes sont supérieures à 1MF...", écrivez le code suivant :

```
FIXER LIMITE RECHERCHE(10)
CHERCHER([Clients];[Clients]Ventes>1000000)
FIXER LIMITE RECHERCHE(0)
```

Exemple 2

Référez-vous au deuxième exemple de la commande **FIXER DESTINATION RECHERCHE**.

FIXER RECHERCHE ET VERROUILLAGE

FIXER RECHERCHE ET VERROUILLAGE (verrou)

Paramètre	Type		Description
verrou	Booléen	⇨	Vrai = verrouiller les enregistrements trouvés par les recherches, Faux = ne pas les verrouiller

Description

La commande **FIXER RECHERCHE ET VERROUILLAGE** vous permet de demander le verrouillage automatique des enregistrements trouvés par toutes les recherches qui suivent son appel dans la transaction courante. Ce mécanisme permet de s'assurer que les enregistrements ne puissent pas être modifiés par un process autre que le process courant entre une recherche et la manipulation des résultats.

Par défaut, les enregistrements trouvés par les recherches ne sont pas verrouillés. Passez **Vrai** dans le paramètre *verrou* pour activer le verrouillage.

Cette commande doit impérativement être utilisée à l'intérieur d'une transaction. Si elle est appelée hors du contexte d'une transaction, une erreur est générée. Ce principe permet un meilleur contrôle du verrouillage des enregistrements. Les enregistrements trouvés restent verrouillés tant que la transaction n'a pas été terminée (qu'elle ait été validée ou annulée). A l'issue de la transaction, tous les enregistrements sont déverrouillés.

Le verrouillage des enregistrements est effectif pour toutes les tables dans la transaction courante.

Lorsqu'une instruction **FIXER RECHERCHE ET VERROUILLAGE(Vrai)** a été exécutée, les commandes de recherche (par exemple **CHERCHER**) adoptent un fonctionnement spécifique si un enregistrement déjà verrouillé est trouvé :

- la recherche est stoppée et la variable système OK prend la valeur 0,
- la sélection courante est vidée,
- l'ensemble système LockedSet contient l'enregistrement verrouillé à l'origine de l'arrêt de la recherche.

Par conséquent, dans ce contexte il est nécessaire de tester l'ensemble LockedSet à l'issue d'une recherche infructueuse (sélection courante vide et/ou variable OK à 0) afin de déterminer la cause de l'échec.

Appelez **FIXER RECHERCHE ET VERROUILLAGE(Faux)** afin de désactiver le mécanisme après usage.

FIXER RECHERCHE ET VERROUILLAGE modifie uniquement le comportement des commandes de recherche, c'est-à-dire :

- **CHERCHER**
- **CHERCHER DANS SELECTION**
- **CHERCHER PAR EXEMPLE**
- **CHERCHER PAR FORMULE**
- **CHERCHER PAR FORMULE DANS SELECTION**
- **CHERCHER PAR SQL**
- **CHERCHER PAR TABLEAU**
- **CHERCHER PAR TABLEAU DANS SELECTION**
- **CHERCHER PAR ATTRIBUT**
- **CHERCHER PAR ATTRIBUT DANS SELECTION**

En revanche, **FIXER RECHERCHE ET VERROUILLAGE** n'affecte pas les autres commandes qui modifient la sélection courante telles que **TOUT SELECTIONNER**, **LIEN RETOUR**, etc.

Exemple

Dans cet exemple, il n'est pas possible de supprimer un client qui aurait été passé de la catégorie "C" à la catégorie "A" par un autre process entre le **CHERCHER** et le **SUPPRIMER SELECTION** :

```
DEBUT TRANSACTION
FIXER RECHERCHE ET VERROUILLAGE(Vrai)
CHERCHER([Clients];[Clients]Catégorie=C)
 `A cet instant, les enregistrements trouvés sont automatiquement verrouillés pour tous les autres process
SUPPRIMER SELECTION([Clients])
FIXER RECHERCHE ET VERROUILLAGE(Faux)
VALIDER TRANSACTION
```

Gestion des erreurs

Si la commande est appelée hors du contexte d'une transaction, une erreur est générée.

Lire dernier chemin recherche

Lire dernier chemin recherche (formatDesc) -> Résultat

Paramètre	Type		Description
formatDesc	Entier long	⇒	Format de description (Texte ou XML)
Résultat	Chaîne	⇐	Description du chemin de la dernière recherche exécutée

Description

La commande **Lire dernier chemin recherche** retourne la description interne détaillée du chemin réel de la dernière recherche effectuée sur les données. Pour plus d'informations sur les descriptions de recherches, reportez-vous à la documentation de la commande **DECRIRE EXECUTION RECHERCHE**.

Cette description est retournée au format Texte ou XML en fonction de la valeur passée dans le paramètre *formatDesc*. Vous pouvez passer une des constantes suivantes, placées dans le thème "**Recherches**" :

Constante	Type	Valeur
Description format Texte	Entier long	0
Description format XML	Entier long	1

Cette commande retourne une valeur significative si la commande **DECRIRE EXECUTION RECHERCHE** a été exécutée au cours de la session.

La description du chemin de la dernière recherche peut être comparée à la description du plan prévu de la dernière recherche (obtenue à l'aide de la commande **Lire dernier plan recherche**) à des fins d'optimisations.

Lire dernier plan recherche

Lire dernier plan recherche (formatDesc) -> Résultat

Paramètre	Type		Description
formatDesc	Entier long	→	Format de description (Texte ou XML)
Résultat	Chaîne	←	Description du plan de la dernière recherche exécutée

Description

La commande **Lire dernier plan recherche** retourne la description interne du plan d'exécution prévu pour la dernière recherche effectuée sur les données. Pour plus d'informations sur les descriptions de recherches, reportez-vous à la commande **DECRIRE EXECUTION RECHERCHE**.

Cette description est retournée au format Texte ou XML en fonction de la valeur passée dans le paramètre *formatDesc*. Vous pouvez passer une des constantes suivantes, placées dans le thème "**Recherches**" :

Constante	Type	Valeur
Description format Texte	Entier long	0
Description format XML	Entier long	1

Cette commande retourne une valeur significative si la commande **DECRIRE EXECUTION RECHERCHE** a été exécutée au cours de la session. La description du plan de la dernière recherche peut être comparée à la description du chemin réel de la dernière recherche (obtenue à l'aide de la commande **Lire dernier chemin recherche**) à des fins d'optimisations.

LIRE DESTINATION RECHERCHE

LIRE DESTINATION RECHERCHE (destinationType ; destinationObjet ; destinationPtr)

Paramètre	Type		Description
destinationType	Entier long	⇐	0 = sélection courante, 1 = ensemble, 2 = sélection temporaire, 3 = variable
destinationObjet	Chaîne	⇐	Nom de l'ensemble ou Nom de la sélection temporaire ou Chaîne vide
destinationPtr	Pointeur	⇐	Pointeur vers variable locale si destinationType = 3

Description

La commande **LIRE DESTINATION RECHERCHE** retourne la destination courante des résultats des recherches pour le process en cours. Par défaut, les résultats des recherches modifient la sélection courante, mais vous pouvez modifier ce fonctionnement l'aide de la commande **FIXER DESTINATION RECHERCHE**.

4D retourne dans le paramètre *destinationType* une valeur indiquant la destination courante des recherches et dans *destinationObjet* le nom de la destination (le cas échéant). Vous pouvez comparer la valeur du paramètre *destinationType* aux constantes du thème **Recherches** :

Constante	Type	Valeur
Vers ensemble	Entier long	1
Vers sélection courante	Entier long	0
Vers sélection temporaire	Entier long	2
Vers variable	Entier long	3

La valeur retournée dans le paramètre *destinationObjet* dépend de la valeur du paramètre *destinationType* :

Paramètre destinationType	Paramètre destinationObjet
0 (sélection courante)	*destinationObjet* est une chaîne vide
1 (ensemble)	*destinationObjet* contient le nom de l'ensemble
2 (sélection temporaire)	*destinationObjet* contient le nom de la sélection temporaire
3 (variable)	*destinationObjet* est une chaîne vide (utiliser le paramètre *destinationPtr*)

Lorsque la destination des recherches est une variable (*destinationType* retourne 3), 4D retourne dans le paramètre *destinationPtr* un pointeur vers cette variable.

Exemple

Nous souhaitons modifier temporairement la destination de recherche, et rétablir ensuite les paramètres précédents :

```
LIRE DESTINATION RECHERCHE($vType;$vNom;$ptr)
 //récupération des paramètres courants
FIXER DESTINATION RECHERCHE(Vers ensemble;"$tempo")
 //modification temporaire de la destination
CHERCHER(...) //recherche
FIXER DESTINATION RECHERCHE($vType;$vNom;$ptr)
 //rétablissement des paramètres
```

Lire limite recherche

Lire limite recherche -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Nombre limite d'enregistrements, 0 = nombre illimité

Description

La commande **Lire limite recherche** retourne la limite du nombre d'enregistrements qu'une recherche pourra trouver dans le process courant.
Vous fixez cette limite à l'aide de la commande **FIXER LIMITE RECHERCHE**.
Par défaut, si aucune limite n'a été définie, la commande retourne 0.

TRIER

TRIER ({laTable ;}{ leChamp }{; > ou < }{; leChamp2 ; > ou <2 ; ... ; leChampN ; > ou <N}{; *})

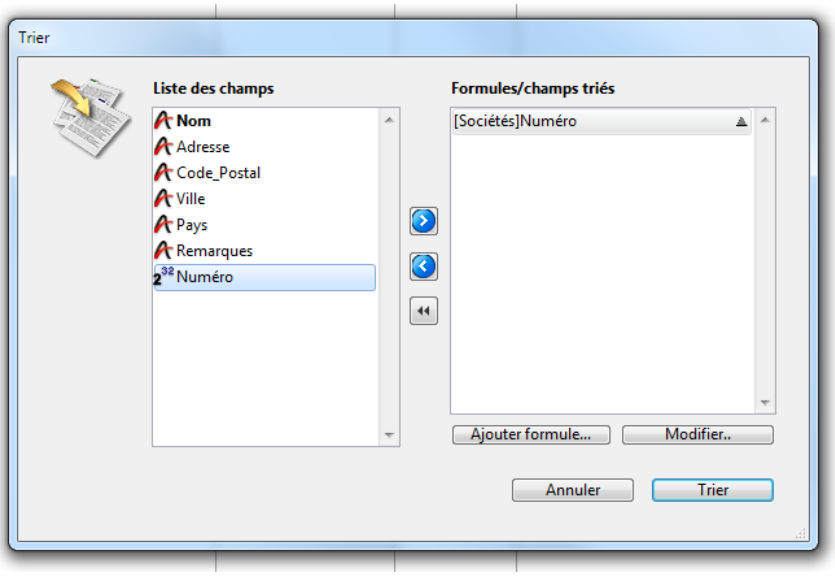
Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle réordonner la sélection courante ou Table par défaut si ce paramètre est omis
leChamp	Champ	⇒	Champ sur lequel effectuer le tri pour chaque niveau
> ou <	Opérateur	⇒	Sens du tri pour chaque niveau : > demander un tri croissant ou < demander un tri décroissant
*	Opérateur	⇒	Attente d'exécution du tri

Description

TRIER trie (réordonne) les enregistrements de la sélection courante de *laTable* pour le process courant. Une fois le tri effectué, le premier enregistrement de la sélection courante devient le nouvel enregistrement courant.

Si vous omettez le paramètre *laTable*, la commande s'applique à la table par défaut, si elle a été définie. Sinon, 4D utilise la table du premier champ passé en paramètre. Si vous ne passez pas de paramètre et si aucune table par défaut n'a été définie, une erreur est retournée.

Si vous ne passez ni le paramètre *leChamp*, ni les paramètres >, < ou *, **TRIER** affiche la boîte de dialogue de l'Editeur de tri de 4D pour *laTable*. Cet éditeur est présenté ci-dessous :



Pour plus d'informations sur l'utilisation de cet éditeur, reportez-vous au manuel *Mode Développement* de 4D.
L'utilisateur construit le tri puis clique sur le bouton **Trier**. Si le tri est correctement effectué, la variable système OK prend la valeur *1*. Si l'utilisateur clique sur **Annuler**, aucun tri n'est effectué et la variable OK prend la valeur *0* (zéro).

Si vous spécifiez les paramètres *leChamp* et *> ou <*, la boîte de dialogue standard de Tri ne s'affiche pas et le tri est entièrement défini par programmation. Vous pouvez trier la sélection courante sur un plusieurs niveaux. Pour chaque niveau de tri, vous passez un champ dans le paramètre *leChamp* et un ordre de tri dans *> ou <*. Si vous passez le paramètre “supérieur à ” (>), l'ordre est croissant. Si vous passez le paramètre “inférieur à ” (<), l'ordre est décroissant. Si vous omettez le paramètre d'ordre *> ou <*, le tri est croissant par défaut.

Si un seul champ est spécifié (tri sur un niveau) et s'il est indexé, le tri tire parti de l'index. Si le champ n'est pas indexé ou si plus d'un champ est utilisé, le tri est effectué de manière séquentielle (hors index composites). Le champ peut appartenir à la table de la sélection que vous triez ou à une table 1 liée à *laTable* par un lien automatique ou manuel. Dans ce cas, le tri est toujours séquentiel.
Si les champs triés sont inclus dans un index composite, **TRIER** tire parti de l'index.

Pour indiquer que le tri ne doit pas être immédiatement effectué, passez en dernier paramètre le symbole *. 4D attendra de rencontrer une nouvelle ligne de tri ne se terminant pas par *pour exécuter le tri. Cette possibilité est utile pour gérer les tris multicritères dans le cadre d'interfaces personnalisées.
Attention : lorsque vous utilisez cette syntaxe, vous ne pouvez passer qu'un seul niveau de tri (un seul champ) par ligne d'instruction.

Quelle que soit la manière dont le tri est défini, si l'opération risque de prendre un certain temps, 4D affiche automatiquement un message contenant un thermomètre de progression. Vous pouvez décider d'afficher ou de ne pas afficher ce message pour le process à l'aide des commandes **SUPPRIMER MESSAGES** et **LAISSER MESSAGES**. Si le thermomètre de progression est affiché, l'utilisateur peut cliquer sur le bouton **Stop** pour interrompre l'opération. Si le tri s'est correctement déroulé, la variable système OK prend la valeur 1. Sinon, si le tri est interrompu, OK prend la valeur 0 (zéro).

Note : Cette commande ne prend pas en charge les champs de type Objet.

Exemple 1

L'exemple suivant affiche la boîte de dialogue de Tri pour la table [Produits] :

```
TRIER([Produits])
```

Exemple 2

L'exemple suivant affiche la boîte de dialogue de Tri pour la table par défaut (si elle a été définie) :

```
TRIER
```

Exemple 3

L'exemple suivant trie la sélection courante de [Produits] par nom dans un ordre croissant :

```
TRIER([Produits];[Produits]Nom;>)
```

Exemple 4

L'exemple suivant trie la sélection courante de [Produits] par nom dans un ordre décroissant :

```
TRIER([Produits];[Produits]Nom;<)
```

Exemple 5

L'exemple suivant trie la sélection courante de [Produits] par type et par prix dans un ordre croissant à chaque niveau :

```
TRIER([Produits];[Produits]Type;>;[Produits]Prix;>)
```

Exemple 6

L'exemple suivant trie la sélection courante de [Produits] par type et par prix dans un ordre décroissant à chaque niveau :

```
TRIER([Produits];[Produits]Type;<;[Produits]Prix;<)
```

Exemple 7

L'exemple suivant trie la sélection courante de [Produits] par type dans un ordre croissant et par prix dans un ordre décroissant :

```
TRIER([Produits];[Produits]Type;>;[Produits]Prix;<)
```

Exemple 8

L'exemple suivant trie la sélection courante de [Produits] par type dans un ordre décroissant et par prix dans un ordre croissant :

```
TRIER([Produits];[Produits]Type;<;[Produits]Prix;>)
```

Exemple 9

L'exemple suivant effectue un tri indexé si le champ [Produits]Nom est indexé :

```
TRIER([Produits];[Produits]Nom;>)
```

Exemple 10

L'exemple suivant trie la sélection courante de [Produits] par nom dans un ordre croissant :

```
TRIER([Produits];[Produits]Nom) //ordre croissant par défaut si > < omis
```

Exemple 11

L'exemple suivant effectue un tri séquentiel, que les champs soient ou non indexés :

```
TRIER([Produits];[Produits]Type;>;[Produits]Prix;>)
```

Exemple 12

L'exemple suivant effectue un tri séquentiel à l'aide d'un champ lié :

```
TRIER([Factures];[Société]Nom;>) ` les factures sont triées par ordre alphabétique sur le champ Nom de la société
```

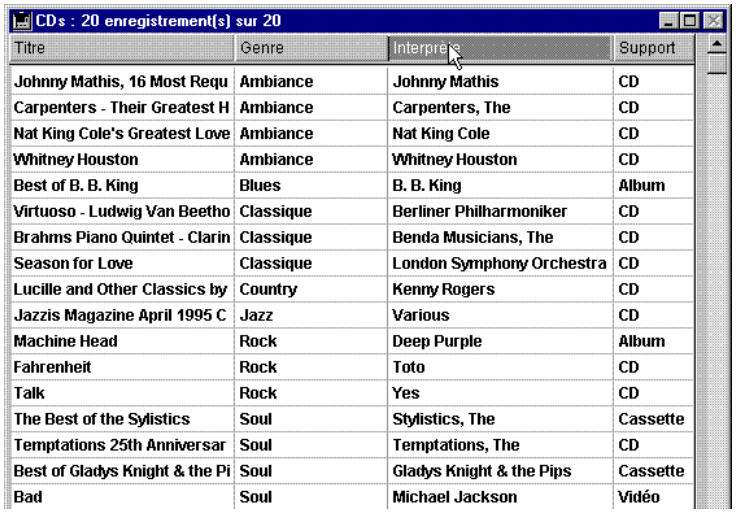
Exemple 13

L'exemple suivant effectue un tri indexé sur deux niveaux si un index composite [Contacts]Nom + [Contacts]Prénom a été défini dans la base :

```
TRIER([Contacts];[Contacts]Nom;>;[Contacts]Prénom;>)
```

Exemple 14

Dans un formulaire sortie affiché en mode Application, vous souhaitez permettre aux utilisateurs de trier une colonne par ordre croissant en cliquant simplement sur son en-tête.
Si l'utilisateur maintient la touche **Maj** enfoncée et clique ensuite sur plusieurs autres colonnes, le tri est multicritères, c'est-à-dire que les colonnes sont triées sur autant de niveaux qu'il y a de clics :



Titre	Genre	Interprète	Support
Johnny Mathis, 16 Most Requ	Ambiance	Johnny Mathis	CD
Carpenters - Their Greatest H	Ambiance	Carpenters, The	CD
Nat King Cole's Greatest Love	Ambiance	Nat King Cole	CD
Whitney Houston	Ambiance	Whitney Houston	CD
Best of B. B. King	Blues	B. B. King	Album
Virtuoso - Ludwig Van Beetho	Classique	Berliner Philharmoniker	CD
Brahms Piano Quintet - Clarin	Classique	Benda Musicians, The	CD
Season for Love	Classique	London Symphony Orchestra	CD
Lucille and Other Classics by	Country	Kenny Rogers	CD
Jazzis Magazine April 1995 C	Jazz	Various	CD
Machine Head	Rock	Deep Purple	Album
Fahrenheit	Rock	Toto	CD
Talk	Rock	Yes	CD
The Best of the Sylistics	Soul	Stylistics, The	Cassette
Temptations 25th Anniversar	Soul	Temptations, The	CD
Best of Gladys Knight & the Pi	Soul	Gladys Knight & the Pips	Cassette
Bad	Soul	Michael Jackson	Vidéo

Chaque en-tête de colonne contient un bouton inversé dont la méthode est du type suivant :

```
MULTITRIS(->[CDs]Titre) //Bouton de l'en-tête de la colonne Titre
```

Chaque bouton appelle la méthode projet ***MULTITRIS*** en passant un pointeur sur le champ de la colonne. Voici le contenu de la méthode projet ***MULTITRIS*** :

```
 // Méthode projet MULTITRIS
 // MULTITRIS (Pointeur)
 // MULTITRIS (->[Table]Champ)

C_POINTEUR($1)
C_ENTIER LONG($nbCrit)

 // Construction des critères
Si(Non(Majuscule enfoncee)) //Si le tri est simple
   TABLEAU POINTEUR(tPtrTriChp;1) //Créons un tableau à 1 élément
   tPtrTriChp{1}:=$1 //Champ sur lequel l'utilisateur a cliqué
Sinon //Si la touche Maj était enfoncée (tri multicritère)
   $nbCrit:=Chercher dans tableau(tPtrTriChp;$1) //Vérifions que le critère n'est pas déjà présent
   Si($nbCrit<0) //Critère inexistant
      INSERER DANS TABLEAU(tPtrTriChp;Taille tableau(tPtrTriChp)+1;1) //Remplissons le tableau
      tPtrTriChp{Taille tableau(tPtrTriChp)}:=$1
   Fin de si
Fin de si
 //Exécution du tri
$nbCrit:=Taille tableau(tPtrTriChp)
Si($nbCrit>0) //S'il y a au moins un élément dans le tableau de pointeurs
   Boucle($i;1;$nbCrit) //Pour chaque critère défini
      TRIER([CDs];(tPtrTriChp{$i})->;>;*) //On construit le tri
   Fin de boucle
   TRIER([CDs]) //Pas de * : on effectue le tri
Fin de si
```

TRIER PAR FORMULE

TRIER PAR FORMULE (laTable ; expression {; > ou <}{; expression2 ; > ou <2 ; ... ; expressionN ; > ou <N})

Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle trier la sélection d'enregistrements
expression	Expression	⇒	Formule de tri des enregistrements (peut être de type Alphanumérique, Réel, Entier, Entier long, Date, Heure ou Booléen)
> ou <	Opérateur	⇒	Ordre de tri pour chaque niveau : > ordre croissant ou < ordre décroissant

Description

TRIER PAR FORMULE trie (réordonne) les enregistrements de la sélection courante de *laTable* pour le process courant sur le critère de tri défini par *expression*. Une fois le tri effectué, le premier enregistrement de la sélection courante devient le nouvel enregistrement courant.

Notez que vous devez spécifier *laTable*. Vous ne pouvez pas utiliser une table par défaut.

Vous pouvez trier la sélection sur un ou plusieurs niveaux. Pour chaque niveau, vous passez une expression dans *expression* et un ordre de tri dans *> ou <*. Si vous passez le symbole “supérieur à” (>), l'ordre est croissant. Si vous passez le symbole “inférieur à” (<), l'ordre est décroissant. Si vous ne passez pas ce paramètre, l'ordre est par défaut croissant.

Le paramètre *expression* peut être de type Alpha, Réel, Entier, Entier long, Date, Heure ou Booléen.

Quelle que soit la manière dont le tri est défini, si l'opération risque de prendre un certain temps, 4D affiche automatiquement un message contenant un thermomètre de progression. Vous pouvez décider d'afficher ou de ne pas afficher ce message pour le process à l'aide des commandes **LAISSER MESSAGES** et **SUPPRIMER MESSAGES**. Si le thermomètre de progression est affiché, l'utilisateur peut cliquer sur le bouton **Stop** pour interrompre l'opération. Si le tri s'est correctement déroulé, la variable système OK prend la valeur 1. Sinon, si le tri est interrompu, OK prend la valeur 0 (zéro).

4D Server : A compter de la version 11 de 4D Server, cette commande est exécutée sur le serveur, ce qui optimise son exécution. A noter qu’en cas d’appel direct de variables dans l'*expression*, le tri est calculé avec la valeur de la variable sur le poste client.
En revanche, ce principe n’est pas appliqué pour les formules utilisant des méthodes qui, elles-mêmes, font appel à des variables (la valeur des variables est évaluée sur le serveur). Dans ce contexte, il peut être judicieux d’utiliser l'attribut de méthode "Exécuter sur serveur" permettant d’exécuter une méthode sur le serveur en lui passant des paramètres (variables) (cf. manuel Mode Développement).
Dans les versions précédentes de 4D Server, cette commande était exécutée sur les postes clients. Par compatibilité, ce fonctionnement est maintenu dans les bases de données converties en version 11. Une préférence de compatibilité et un sélecteur de la commande **FIXER PARAMETRE BASE** permettent toutefois d'adopter le fonctionnement de la version 11 (exécution sur le serveur) dans ces bases de données.

Exemple

L'exemple suivant trie les enregistrements de la table [Personnes] dans l'ordre décroissant par rapport à la longueur du nom de famille de chaque personne. L'enregistrement de la personne qui a le nom le plus long sera le premier enregistrement de la sélection courante :

```
TRIER PAR FORMULE([Personnes];Longueur([Personnes]Nom);<)
```

Trouver dans champ

Trouver dans champ (champCible ; valeur) -> Résultat

Paramètre	Type		Description
champCible	Champ	⇒	Champ sur lequel effectuer la recherche
valeur	Champ, Variable	⇒	Valeur à rechercher
		⇐	Valeur trouvée
Résultat	Entier long	⇒	Numéro de l'enregistrement trouvé ou -1 si pas d'enregistrement trouvé

Description

La commande **Trouver dans champ** retourne le numéro du premier enregistrement dont le champ *champCible* est égal à la valeur *valeur*.
Si aucun enregistrement ne correspond au critère, **Trouver dans champ** retourne -1.

Après l'appel, le paramètre *valeur* contient la valeur effectivement trouvée. Ce fonctionnement permet d'effectuer des recherches utilisant le caractère "@" sur des champs de type alpha, et pour lesquelles il est nécessaire de récupérer la valeur trouvée.

Note : Du fait de ce principe, vous ne pouvez pas utiliser un *paramètre* ($1, $2...) dans *valeur* car cela entraînerait des dysfonctionnements en mode compilé. De même, si vous passez un champ dans le paramètre *valeur*, gardez à l'esprit que sa valeur sera réaffectée si la recherche aboutit (la commande **Enregistrement modifie**, notamment, retournera Vrai pour l'enregistrement courant de la table).

La commande ne modifie ni la sélection courante, ni l'enregistrement courant.

Cette fonction, très rapide, est particulièrement utile pour prévenir la création de doublons au moment de la saisie de données.

Note historique : Dans les anciennes versions de 4D, la commande **Trouver dans champ** était nommée **Trouver clef index** et ne fonctionnait qu'avec les champs indexés. La commande a été renommée et la limitation supprimée à compter de 4D v11 SQL.

Exemple 1

Dans une base de données de CD audio, vous souhaitez vérifier, au moment de la saisie d'un nouveau nom de chanteur, si celui-ci n'existe pas déjà dans la base. Comme il peut exister des homonymes, vous ne souhaitez pas toutefois que le champ [Chanteur]Nom soit unique. Pour cela, dans le formulaire d'entrée, vous écrivez dans la méthode objet du champ [Chanteur]Nom :

```
Si(Evenement formulaire=Sur données modifiées)
   $EnrgNum:=Trouver dans champ([Chanteur]Nom;[Chanteur]Nom)
   Si($EnrgNum #-1) ` Si ce nom a déjà été saisi
      CONFIRMER("Un chanteur de ce nom existe déjà. Voulez-vous visualiser sa fiche ?";"Oui";"Non")
      Si(OK=1)
         ALLER A ENREGISTREMENT([Chanteur];$EnrgNum)
      Fin de si
   Fin de si
Fin de si
```

Exemple 2

Voici un exemple permettant de vérifier l'existence d'une valeur :

```
C_ENTIER LONG($id;$1)
$id:=$1
Si(Trouver dans champ([MaTable]MonID;$id)>=0)
   $0:=Vrai
Sinon
   $0:=Faux
Fin de si
```

Remarquez le >= qui permet de couvrir tous les cas. En effet, la fonction retourne un numéro d'enregistrement et le premier enregistrement porte le numéro 0.

Ressources

- Ressources
- FERMER FICHIER RESSOURCES
- Lire chaine dans liste
- Lire nom ressource
- Lire proprietes ressource
- LIRE RESSOURCE
- Lire ressource chaine
- LIRE RESSOURCE ICONE
- LIRE RESSOURCE IMAGE
- Lire ressource texte
- LISTE DE CHAINES VERS TABLEAU
- LISTE RESSOURCES
- LISTE TYPES RESSOURCE
- Ouvrir fichier ressources
- *_o_Créer fichier ressources*
- *_o_ÉCRIRE NOM RESSOURCE*
- *_o_ÉCRIRE PROPRIÉTÉS RESSOURCE*
- *_o_ÉCRIRE RESSOURCE*
- *_o_ÉCRIRE RESSOURCE CHAINE*
- *_o_ÉCRIRE RESSOURCE IMAGE*
- *_o_ÉCRIRE RESSOURCE TEXTE*
- *_o_Lire ID ressource composant*
- *_o_SUPPRIMER RESSOURCE*
- *_o_TABLEAU VERS LISTE DE CHAÎNES*

Notes de compatibilité sur l'écriture des ressources (4D v13)

Comme annoncé depuis 4D v11 (cf. paragraphe ci-dessous), les mécanismes basés sur l'utilisation de fichiers de ressources sont obsolètes. **Les commandes du thème "Ressources" permettant d'écrire dans des fichiers de ressources ne doivent plus être utilisées, elles seront supprimées dans les versions prochaines de 4D.** Les commandes permettant de lire des ressources sont maintenues par compatibilité.

Notes de compatibilité sur les mécanismes de gestion des ressources (4D v11)

La gestion des ressources a été modifiée dans 4D à compter de la v11. Conformément aux orientations définies par Apple et mises en oeuvres dans les versions de Mac OS les plus récentes, le concept de ressources au sens strict (cf. définition ci-dessous) est désormais obsolète et va être progressivement abandonné. De nouveaux mécanismes sont mis en place pour prendre en charge les besoins précédemment couverts par les ressources : fichiers XLIFF pour la traduction des chaînes de caractères, fichiers images .png... De fait, les fichiers de ressources disparaissent au profit de fichiers de type standard. 4D accompagne cette évolution et, à partir de la v11, propose de nouveaux outils pour la gestion des traductions des bases de données, tout en maintenant la compatibilité avec les systèmes existants.

Compatibilité

Pour le maintien de la compatibilité et afin de permettre l'adaptation progressive des applications existantes, les mécanismes de gestion des ressources continuent de fonctionner dans 4D v11, à quelques différences près :

- Lorsqu'ils sont présents, les fichiers de ressources sont toujours pris en charge par 4D et le principe de la "chaîne des fichiers de ressources" (ouverture successive de plusieurs fichiers de ressources) reste valide. La "chaîne des fichiers de ressources" comprend notamment les fichiers .rsr ou .4dr. des bases de données converties (ouverts automatiquement) et les fichiers de ressources personnalisés ouverts via les commandes de ce thème.
- Toutefois, pour cause d'évolution de l'architecture interne, il n'est plus possible d'accéder directement via ces commandes (ou via les références dynamiques) aux ressources de l'application 4D ni à celles du système. Certains développeurs ont pu tirer parti de ressources internes de 4D pour leurs interfaces (par exemple les ressources contenant les noms des mois ou ceux des commandes du langage). Cette pratique est désormais à proscrire. Dans la plupart des cas, il est possible d'utiliser d'autres moyens que les ressources internes de 4D (constantes, commandes du langage...). Afin de limiter l'impact de cette modification sur les bases existantes, un système de substitution a été mis en place, basé sur l'externalisation des ressources les plus utilisées. Il est cependant fortement conseillé de faire évoluer les bases de données converties et de supprimer les appels aux ressources internes de 4D.
- Les bases de données créées avec 4D à partir de la v11 ne comportent plus par défaut de fichiers .RSR (ressources de la structure) et .4DR (ressources des données).

Principes actuels de gestion des ressources

Depuis 4D v11, la notion de "ressources" doit être entendue au sens large comme "fichiers nécessaires à la traduction de l'interface des applications". L'architecture actuelle des ressources s'appuie sur un dossier, nommé **Resources**, impérativement situé à côté du fichier de structure de la base (.4db ou .4dc). Vous devez placer dans ce dossier tous les fichiers nécessaires à la traduction ou la personnalisation de l'interface de l'application (fichiers image, texte, XLIFF...).
Il peut également contenir les éventuels fichiers de resources "ancienne génération" de la base (fichiers .rsr). Attention, ces fichiers ne sont pas automatiquement inclus dans la chaîne des ressources, ils devront être ouverts via les commandes standard de manipulation des ressources de 4D. 4D utilise des mécanismes automatiques pour l'exploitation du contenu de ce dossier, notamment pour la gestion des fichiers XLIFF (ce point est traité dans le manuel *Mode Développement*). Par compatibilité, les commandes **Lire chaine dans liste** et **LISTE DE CHAINES VERS TABLEAU** du thème "Ressources" permettent de tirer parti de cette architecture, mais il est désormais conseillé d'utiliser la commande **Lire traduction chaine** du thème "Chaînes de caractères".

FERMER FICHIER RESSOURCES

FERMER FICHIER RESSOURCES (resFichier)

Paramètre	Type		Description
resFichier	RefDoc	⇨	Numéro de référence de fichier de ressources

Description

La commande **FERMER FICHIER RESSOURCES** referme le fichier de ressources dont vous avez passé le numéro de référence dans *resFichier*.

Même si vous avez ouvert plusieurs fois un fichier de ressources, il vous suffit d'appeler **FERMER FICHIER RESSOURCES** une seule fois pour le refermer.

Si vous appliquez **FERMER FICHIER RESSOURCES** au fichier de ressources de l'application 4D ou de la base, la commande le détecte et ne fait rien.

Si vous passez un numéro de référence de fichier de ressources non valide, la commande ne fait rien.

N'oubliez pas d'appeler finalement **FERMER FICHIER RESSOURCES** pour un fichier de ressources que vous avez ouvert à l'aide de la commande **Ouvrir fichier ressources**. Notez cependant que 4D referme automatiquement tous les fichiers de ressources ouverts lorsque vous quittez l'application ou ouvrez une autre base de données.

Lire chaine dans liste

Lire chaine dans liste (resNum ; strNum {; resFichier}) -> Résultat

Paramètre	Type		Description
resNum	Entier long	⇒	Numéro de ressource ou Attribut 'id' de l'élément 'group' (XLIFF)
strNum	Entier long	⇒	Numéro de chaîne ou Attribut 'id' de l'élément 'trans-unit' (XLIFF)
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources Si omis : tous les fichiers XLIFF ou les fichiers de ressources ouverts
Résultat	Chaîne	⇐	Valeur de la chaîne indexée

Description

La commande **Lire chaine dans liste** retourne :

- soit une des chaînes stockées dans la ressource liste de chaînes ("STR#") dont vous avez passé le numéro d'ID dans *resNum*,
- soit une chaîne stockée dans un fichier XLIFF ouvert dont vous avez passé l'attribut 'id' de l'élément 'group' dans *resNum* (cf. ci-dessous "Compatibilité avec l'architecture XLIFF").

Vous passez le numéro de la chaîne dans *strNum*. Les chaînes d'une ressource liste de chaînes sont numérotées de 1 à N. Pour récupérer toutes les chaînes (et donc leur nombre) d'une ressource liste de chaînes, utilisez la commande **LISTE DE CHAINES VERS TABLEAU**.

Si la ressource n'est pas trouvée, ou si la chaîne n'est pas trouvée à l'intérieur de la ressource, une chaîne vide est retournée et la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans *resFichier*, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre *resFichier*, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Note : Chaque chaîne d'une ressource liste de chaînes peut contenir jusqu'à 255 caractères.

Compatibilité avec l'architecture XLIFF

La commande **Lire chaine dans liste** est compatible avec l'architecture XLIFF de 4D à compter de la v11 : la commande recherche dans un premier temps les valeurs correspondant à *resNum* et *strNum* dans tous les fichiers XLIFF ouverts (si le paramètre *resFichier* est omis). Dans ce cas, *resNum* désigne l'attribut **id** de l'élément **group** et *strNum* désigne l'attribut **id** de l'élément **trans-unit**. Si la valeur n'est pas trouvée, la commande poursuit la recherche dans les fichiers de ressources ouverts. Pour plus d'informations sur l'architecture XLIFF dans 4D, reportez-vous au manuel Mode Développement.

Variables et ensembles système

OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

Lire nom ressource

Lire nom ressource (resType ; resNum {; resFichier}) -> Résultat

Paramètre	Type		Description
resType	Chaîne	⇒	Type de ressource (4 caractères)
resNum	Entier long	⇒	Numéro de référence de ressource (ID)
resFichier	RefDoc	⇒	Numéro de référence du fichier de ressource ou Tous les fichiers de ressources ouverts si omis
Résultat	Chaîne	⇐	Nom de la ressource

Description

Lire nom ressource retourne le nom de la ressource dont le type est passé dans *resType* et le numéro de référence (ID) dans *resNum*.

Si vous ne passez pas le paramètre *resFichier*, la ressource est recherchée dans tous les fichiers de ressources ouverts. Si vous passez un numéro de référence de fichier de ressource dans le paramètre *resFichier*, la ressource n'est recherchée que dans ce fichier.

Si la ressource n'existe pas, **Lire nom ressource** retourne une chaîne vide.

Lire proprietes ressource

Lire proprietes ressource (resType ; resNum {; resFichier}) -> Résultat

Paramètre	Type		Description
resType	Chaîne	⇒	Type de ressource (4 caractères)
resNum	Entier long	⇒	Numéro de référence de ressource (ID)
resFichier	RefDoc	⇒	Numéro de référence du fichier de ressource ou Tous les fichiers de ressources ouverts si omis
Résultat	Entier long	⇐	Attributs de la ressource

Description

Lire proprietes ressource retourne les attributs de la ressource dont le type est passé dans le paramètre *resType* et le numéro de référence dans *resNum*.

Si vous ne passez pas le paramètre *resFichier*, la ressource est recherchée dans les fichiers de ressources ouverts. Si vous passez un numéro de référence de fichier de ressource dans le paramètre *resFichier*, la ressource n'est recherchée que dans ce fichier.

Si la ressource n'existe pas, **Lire proprietes ressource** retourne *0* (zéro) et la variable OK prend également la valeur *0* (zéro).

La valeur numérique retournée par **Lire proprietes ressource** doit être considérée comme une valeur binaire dont chaque bit a une signification particulière. Pour une description des attributs des ressources et leurs effets, référez-vous à la commande **_o_ÉCRIRE PROPRIÉTÉS RESSOURCE**.

Exemple

Référez-vous à l'exemple de la commande **Lire nom ressource**.

Variables et ensembles système

La variable OK prend la valeur 0 si la ressource n'existe pas, sinon elle prend la valeur 1.

LIRE RESSOURCE

LIRE RESSOURCE (resType ; resNum ; resDonnées {; resFichier})

Paramètre	Type		Description
resType	Chaîne	⇒	Type de ressource (4 caractères)
resNum	Entier long	⇒	Numéro de ressource
resDonnées	BLOB	⇒	Champ ou variable BLOB devant recevoir les données
		⇐	Contenu de la ressource
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou Tous les fichiers de ressources ouverts si omis

Description

La commande **LIRE RESSOURCE** retourne dans le champ ou la variable BLOB *resDonnées* le contenu de la ressource dont le type et le numéro sont passés dans *resType* et *resNum*.

Important : Vous devez passer une chaîne de 4 caractères dans *resType*.

Si la ressource n'est pas trouvée, le paramètre *resDonnées* est laissé inchangé et la variable OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans *resFichier*, la ressource sera recherchée dans ce fichier seulement. Si ne passez pas le paramètre *resFichier*, la première occurrence de la ressource trouvée en remontant la chaîne des fichiers de ressources sera retournée.

Note : La taille d'une ressource peut atteindre plusieurs méga-octets.

Indépendance de plate-forme

Rappelez-vous que vous travaillez avec des ressources issues de Mac OS. Quelle que soit la plate-forme utilisée, les valeurs internes des ressources comme des entiers longs sont stockées avec l'ordre d'octets ("byte ordering") Mac OS. Sous Windows, pour les données des ressources standard (telles que les ressources listes de chaînes et les ressources images) l'ordre des octets est automatiquement inversé ("byte swapping") si nécessaire. D'un autre côté, si vous créez et utilisez vos propres structures internes de données, c'est à vous d'effectuer l'inversion d'octets des données lorsque vous les extrayez d'un BLOB (par exemple en passant Ordre octets Macintosh à une commande telle que **BLOB vers entier long**).

Exemple

Reportez-vous à l'exemple de la commande **_o_ÉCRIRE RESSOURCE**.

Variables et ensembles système

Si la ressource est trouvée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 (zéro).

Gestion des erreurs

S'il n'y a pas assez de mémoire disponible pour charger l'image, une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

Lire ressource chaine

Lire ressource chaine (resNum {; resFichier}) -> Résultat

Paramètre	Type		Description
resNum	Entier long	⇒	Numéro de ressource
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis
Résultat	Chaîne	⇐	Contenu de la ressource STR

Description

La commande **Lire ressource chaine** retourne la chaîne stockée dans la ressource chaîne ("STR ") dont vous avez passé le numéro d'ID dans *resNum*.

Si la ressource n'est pas trouvée, une chaîne vide est retournée et la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans *resFichier*, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre *resFichier*, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Note : Une ressource chaîne peut contenir jusqu'à 255 caractères.

Exemple

L'exemple suivant affiche le contenu de la ressource chaîne d'ID=20911 qui doit se trouver dans au moins un des fichiers de ressources ouverts :

```
ALERTE(Lire ressource chaine(20911))
```

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

LIRE RESSOURCE ICONE

LIRE RESSOURCE ICONE (resNum ; resDonnées {; resFichier})

Paramètre	Type		Description
resNum	Entier long	⇨	Numéro de ressource icône
resDonnées	Image	⇨	Champ ou variable image devant recevoir l'icône
		⇦	Image résultante
resFichier	RefDoc	⇨	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis

Note de compatibilité

Cette commande n'est pas prise en charge dans les versions 64 bits de 4D. Elle retourne une erreur lorsqu'elle est exécutée dans cet environnement.

Description

La commande **LIRE RESSOURCE ICONE** retourne dans le champ ou la variable image *resDonnées* l'icône stockée dans la ressource icône couleur ("cicn") dont vous avez passé le numéro d'ID dans *resNum*.

Si la ressource n'est pas trouvée, le paramètre *resDonnées* reste inchangé et la variable système OK prend la valeur 0 (zéro).

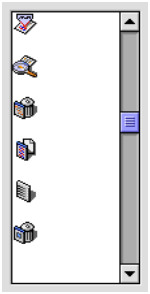
Si vous passez un numéro de référence de fichier de ressources valide dans *resFichier*, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre *resFichier*, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Exemple

L'exemple suivant charge dans un tableau image les icônes couleur situées dans l'application 4D en cours d'utilisation :

```
Si(Sous Windows)
   $vh4DResFile:=Ouvrir fichier ressources(Remplacer chaine(Fichier application;".EXE";".RSR"))
Sinon
   $vh4DResFile:=Ouvrir fichier ressources(Fichier application)
Fin de si
LISTE RESSOURCES("cicn";$alResID;$asResNom;$vh4DResFile)
$vlNbIcons:=Taille tableau($alResID)
TABLEAU IMAGE(ag4DIcon;$vlNbIcons)
Boucle($vlElem;1;$vlNbIcons)
   LIRE RESSOURCE ICONE($alResID{$vlElem};ag4DIcon{$vlElem};$vh4DResFile)
Fin de boucle
```

Une fois ce code exécuté, le tableau aura l'aspect suivant lorsqu'il sera affiché dans un formulaire :



Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

LIRE RESSOURCE IMAGE

LIRE RESSOURCE IMAGE (resNum ; resDonnées {; resFichier})

Paramètre	Type		Description
resNum	Entier long	⇒	Numéro de ressource
resDonnées	Champ, Variable	⇒	Champ ou variable image devant recevoir l'image
		⇐	Contenu de la ressource PICT
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis

Description

La commande **LIRE RESSOURCE IMAGE** retourne dans le champ ou la variable image désigné(e) par *resDonnées* l'image stockée dans la ressource image ("PICT") dont vous passé le numéro dans *resNum*.

Si la ressource n'est pas trouvée, *resDonnées* n'est pas modifié et la variable OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans *resFichier*, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre *resFichier*, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Note : La taille d'une ressource image peut atteindre plusieurs méga-octets, voire davantage.

Exemple

Reportez-vous à l'exemple de la commande **LISTE RESSOURCES**.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

Gestion des erreurs

S'il n'y a pas assez de mémoire disponible pour charger l'image, une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

Lire ressource texte

Lire ressource texte (resNum {; resFichier}) -> Résultat

Paramètre	Type		Description
resNum	Entier long	⇒	Numéro de ressource
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis
Résultat	Texte	⇐	Contenu de la ressource TEXT

Description

La commande **Lire ressource texte** retourne le texte stocké dans la ressource texte ("TEXT") dont vous avez passé le numéro d'ID dans *resNum*.

Si la ressource n'est pas trouvée, un texte vide est retourné et la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans *resFichier*, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre *resFichier*, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Note : Une ressource texte peut contenir jusqu'à 32 000 caractères.

Exemple

L'exemple suivant affiche le contenu de la ressource texte d'ID=20800 qui doit se trouver dans au moins un des fichiers de ressources ouverts :

```
ALERTE(Lire ressource texte(20800))
```

Variables et ensembles système

OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

LISTE DE CHAINES VERS TABLEAU

LISTE DE CHAINES VERS TABLEAU (resNum ; tabChaînes {; resFichier})

Paramètre	Type		Description
resNum	Entier long	⇒	Numéro de ressource ou Attribut 'id' de l'élément 'group' (XLIFF)
tabChaînes	Tableau chaîne	⇐	Chaînes de la ressource STR# ou Chaînes de l'élément 'group' (XLIFF)
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources Si omis : tous les fichiers XLIFF ou les fichiers de ressources ouverts

Description

La commande **LISTE DE CHAINES VERS TABLEAU** remplit le tableau *chaînes* avec :

- soit les chaînes stockées dans la ressource de type liste de chaînes ("STR#") dont vous avez passé le numéro d'ID dans *resNum*.
- soit une chaîne stockée dans un fichier XLIFF ouvert dont vous avez passé l'attribut 'id' de l'élément 'group' dans *resNum* (cf. ci-dessous "Compatibilité avec l'architecture XLIFF").

Si la ressource n'est pas trouvée, le tableau *chaînes* reste inchangé et la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans *resFichier*, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre *resFichier*, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Si vous ne pré-déclarez pas le tableau *chaînes* avant d'appeler **LISTE DE CHAINES VERS TABLEAU**, la commande crée un tableau de type Texte. Si vous pré-déclarez le tableau, vous pouvez lui assigner le type Alpha ou Texte.

Note : Chaque chaîne d'une ressource liste de chaînes peut contenir jusqu'à 255 caractères.

Conseil : Lorsque vous utilisez des ressources listes de chaînes, limitez-vous à des ressources de 32 Ko maximum et quelques centaines de chaînes par ressource.

Compatibilité avec l'architecture XLIFF

La commande **LISTE DE CHAINES VERS TABLEAU** est compatible avec l'architecture XLIFF de 4D v11 : la commande recherche dans un premier temps la valeur correspondant à *resNum* dans tous les fichiers XLIFF ouverts (si le paramètre *resFichier* est omis) et remplit le tableau *chaînes* avec les valeurs correspondantes. Dans ce cas, *resNum* désigne l'attribut **id** de l'élément **group** et le tableau *chaînes* contient toutes les chaînes de l'élément. Si la valeur n'est pas trouvée, la commande poursuit la recherche dans les fichiers de ressources ouverts.
Pour plus d'informations sur l'architecture XLIFF dans 4D, reportez-vous au manuel Mode Développement.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

LISTE RESSOURCES

LISTE RESSOURCES (resType ; resNums ; resNoms {; resFichier})

Paramètre	Type		Description
resType	Chaîne	⇒	Type de ressource (4 caractères)
resNums	Tableau entier long	⇐	Numéros des ressources de ce type
resNoms	Tableau chaîne	⇐	Noms des ressources de ce type
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis

Description

La commande **LISTE RESSOURCES** remplit les tableaux *resNums* et *resNoms* avec les numéros et les noms des ressources dont vous avez passé le type dans *resType*.

Important : Vous devez passez dans *resType* une chaîne de 4 caractères.

Si vous passez un numéro de référence de fichier de ressources valide dans le paramètre optionnel *resFichier*, seules les ressources présentes dans ce fichier seront listées. Si vous ne passez pas le paramètre *resFichier*, toutes les ressources de tous les fichiers de ressources ouverts seront listées.

Si vous ne pré-déclarez pas les tableaux *resNums* et *resNoms* avant d'appeler **LISTE RESSOURCES**, la commande créera par défaut le tableau *resNums* avec le type Entier long et *resNoms* avec le type Texte. Si vous pré-déclarez les tableaux, vous devez attribuer le type Entier long à *resNums*, mais pouvez attribuer le type Alpha ou Texte à *resNoms*.

Après l'appel, vous pouvez tester le nombre de ressources qui ont été trouvées en appliquant la commande **Taille tableau** au tableau *resNums* ou *resNoms*.

Exemple 1

L'exemple suivant remplit les tableaux *$alResNum* et *$atResNom* avec les numéros et les noms des ressources de type Listes de chaînes présentes dans le fichier de structure de la base :

```
Si(Sous Windows)
   $vhStructureResFile:=Ouvrir fichier ressources(Remplacer chaine(Fichier structure;".4DB";".RSR"))
Sinon
   $vhStructureResFile:=Ouvrir fichier ressources(Fichier structure)
Fin de si
Si(OK=1)
   LISTE RESSOURCES("STR#";$alResNum;$atResNom;$vhStructureResFile)
Fin de si
```

Exemple 2

L'exemple suivant copie dans la bibliothèque d'images de la base les ressources image présentes dans tous les fichiers de ressources ouverts :

```
LISTE RESSOURCES("PICT";$alResNum;$atResNom)
Creer fenetre(50;50;550;120;5;"Copie des ressources PICT...")
Boucle($vlElem;1;Taille tableau($alResNum))
   LIRE RESSOURCE IMAGE($alResNum{$vlElem};$vgImage)
   Si(OK=1)
      $vsNom:=$atResNom{$vlElem}
      Si($vsNom="")
         $vsNom:="PICT resID="+Chaine($alResNum{$vlElem})
      Fin de si
      EFFACER FENETRE
      POSITION MESSAGE(2;1)
      MESSAGE("Ajout de l'image ^""+$vsNom+"" à la bibliothèque d'images de la base.")
      ECRIRE IMAGE DANS BIBLIOTHEQUE($vgImage;$alResNum{$vlElem};$vsNom)
   Fin de si
Fin de boucle
FERMER FENETRE
```

LISTE TYPES RESSOURCE

LISTE TYPES RESSOURCE (resTypes {; resFichier})

Paramètre	Type		Description
resTypes	Tableau chaîne	⇐	Liste des types de ressources disponibles
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts (si ce paramètre est omis)

Description

La commande **LISTE TYPES RESSOURCE** remplit le tableau *resTypes* avec les types des ressources présentes dans le(s) fichier(s) de resources ouvert(s).

Si vous passez un numéro de référence de fichier de ressources valide dans le paramètre optionnel *resFichier*, seules les ressources présentes dans ce fichier seront listées. Si vous ne passez pas le paramètre *resFichier*, toutes les ressources de tous les fichiers de ressources ouverts seront listées.

Si vous ne pré-déclarez pas le tableau *resTypes* avant d'appeler **LISTE TYPES RESSOURCE**, la commande créera par défaut un tableau de type Texte. Si vous pré-déclarez le tableau, vous pouvez lui attribuer le type Alpha ou Texte.

Après l'appel, vous pouvez tester le nombre de types de ressources différents qui ont été trouvés en appliquant la commande **Taille tableau** au tableau *resTypes*.

Exemple 1

L'exemple suivant remplit le tableau *atResType* avec les types de ressources présents dans tous les fichiers de ressource ouverts :

```
LISTE TYPES RESSOURCE(atResType)
```

Exemple 2

L'exemple suivant vous indique si le fichier de structure Mac OS que vous utilisez contient des plug-ins 4D "ancien modèle", qui devront être mis à jour si vous voulez exploiter la base sous Windows :

```
$vhResFile:=Ouvrir fichier ressources(Fichier structure)
LISTE TYPES RESSOURCE(atResType;$vhResFile)
Si(Chercher dans tableau(atResType;"4DEX")>0)
   ALERTE("Cette base contient des plug-ins 4D basés sur l'ancien système."+(Caractere(13)*2)+
   "Vous devrez les mettre à jour pour pouvoir utiliser la base sous Windows.")
Fin de si
```

Note : Le fichier de structure n'est pas le seul fichier dans lequel des plug-ins "ancien modèle" ont pu être installés. La base peut également être associée à un fichier "Routines Externes" ou "Proc.Ext".

Exemple 3

La méthode projet suivante retourne le nombre de ressources présentes dans un fichier de ressources :

```
  ` Méthode projet Compter ressources
  ` Compter ressources ( Heure ) -> Entier long
  ` Compter ressources ( DocRef ) -> Nombre de ressources

C_ENTIER LONG($0)
C_HEURE($1)

$0:=0
LISTE TYPES RESSOURCE($atResType;$1)
Boucle($vlElem;1;Taille tableau($atResType))
   LISTE RESSOURCES($atResType{$vlElem};$alResID;$atResName;$1)
   $0:=$0+Taille tableau($alResID)
Fin de boucle
```

Une fois que cette méthode est implémentée dans votre base, vous pouvez écrire par exemple :

```
$vhResFile:=Ouvrir fichier ressources("")
Si(OK=1)
   ALERTE("Le fichier ^"+Document+"" contient "+Chaine(Compter ressources($vhResFile))+
   " ressource(s).")
   FERMER FICHIER RESSOURCES($vhResFile)
Fin de si
```

Ouvrir fichier ressources

Ouvrir fichier ressources (resNomFichier {; typeFichier}) -> Résultat

Paramètre	Type		Description
resNomFichier	Chaîne	➡	Nom ou chemin d'accès complet du fichier de ressources ou chaîne vide pour afficher la boîte de dialogue standard d'ouverture de fichiers
typeFichier	Chaîne	➡	Type de fichier Mac OS (chaîne de 4 caractères), ou extension de fichier Windows (chaîne de 1 à 3 caractères), ou fichier de ressources ("res " / .RES) si omis
Résultat	RefDoc	⬅	Numéro de référence du fichier de ressources

Description

La commande **Ouvrir fichier ressources** ouvre le fichier de ressources dont vous avez passé le nom ou le chemin d'accès complet dans le paramètre *resNomFichier*.

Si vous passez un nom de fichier, celui-ci doit se trouver dans le même dossier/répertoire que le fichier de structure de la base. Pour ouvrir un fichier de ressources se trouvant dans un autre dossier, passez un chemin d'accès complet dans *resNomFichier*.

Si vous passez une chaîne vide dans *resNomFichier*, la boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de sélectionner le fichier à ouvrir. Si l'utilisateur clique sur **Annuler** dans cette boîte de dialogue, aucun fichier de ressources n'est ouvert, **Ouvrir fichier ressources** retourne une valeur nulle dans *RefDoc* et la variable OK prend la valeur 0.

Par défaut, la commande ouvre la resource fork du fichier passé en paramètre. Si celle-ci est vide, la commande ouvre la data fork — si elle contient des ressources. Pour plus d'informations sur ce point, reportez-vous à la section **Date du jour**.

Si le fichier de ressources est correctement ouvert, **Ouvrir fichier ressources** retourne son numéro de référence de fichier et met la variable OK à 1. Si le fichier de ressources n'existe pas ou si le fichier de que vous tentez d'ouvrir n'est pas un fichier de ressources, une erreur est générée.

- Sous Mac OS, si vous utilisez la boîte de dialogue standard d'ouverture de fichiers, tous les fichiers sont présentés par défaut. Pour ne faire apparaître que les fichiers d'un type particulier, spécifiez-le dans le paramètre optionnel *typeFichier*.
- Sous Windows, si vous utilisez la boîte de dialogue standard d'ouverture de fichiers, tous les fichiers sont présentés par défaut. Pour ne faire apparaître que les fichiers d'un type particulier, passez dans *typeFichier* une extension de fichier Windows de 1 à 3 caractères ou un type de fichier Macintosh associé à une extension Windows à l'aide de la commande **ASSOCIER TYPES FICHIER**.

N'oubliez pas d'appeler finalement **FERMER FICHIER RESSOURCES** pour le fichier de ressources. Notez cependant que 4D referme automatiquement tous les fichiers de ressources ouverts par l'intermédiaire de **Ouvrir fichier ressources** lorsque vous quittez l'application ou ouvrez une autre base de données.

A la différence de la commande **Ouvrir document** qui ouvre par défaut un document avec un accès exclusif en lecture-écriture, **Ouvrir fichier ressources** vous permet d'ouvrir un fichier de ressources déjà ouvert dans la session 4D. Par exemple, lorsque vous tentez d'ouvrir deux fois le même document avec **Ouvrir document**, une erreur d'E/S vous est retournée lors de la seconde opération. En revanche, vous pouvez accéder à un fichier de ressources déjà ouvert lors de la session 4D : **Ouvrir fichier ressources** retourne son numéro de référence. Même lorsque vous ouvrez plusieurs fois un fichier de ressources, il vous suffit d'appeler **FERMER FICHIER RESSOURCES** une seule fois pour refermer ce fichier. Notez que ce fonctionnement n'est valable que lorsque le fichier de ressources est ouvert à l'intérieur de la session 4D. Si vous tentez d'ouvrir un fichier de ressources déjà ouvert par une autre application, une erreur d'E/S vous sera retournée.

ATTENTION :

- Il est interdit d'accéder aux fichiers de ressources des applications 4D et des bases fusionnées avec 4D Desktop.
- Bien que techniquement possible, l'accès au fichier de ressources de la structure de la base est fortement déconseillé car ce fonctionnement devient caduc lorsque la base est compilée et fusionnée avec 4D Desktop.
  Si toutefois vous accédez au fichier de ressources de la structure et souhaitez ajouter, supprimer ou modifier des ressources par programmation, pensez à tester l'environnement dans lequel la base s'exécute. Avec 4D Server, cela posera certainement d'épineux problèmes. Si, par exemple, vous modifiez une ressource sur le poste serveur (via une méthode base ou une procédure stockée), vous allez en définitive perturber le système d'administration de 4D Server chargé de distribuer de manière transparente les ressources aux postes clients. Notez qu'avec 4D Client vous n'accédez pas directement au fichier de structure : il est situé sur le poste serveur.
- Pour toutes ces raisons, si vous exploitez des ressources, vous devez les stocker dans vos propres fichiers.
- Lorsque vous travaillez avec vos propres ressources, n'utilisez pas de numéros de ressources négatifs, ils sont réservés au Système d'exploitation. N'utilisez pas non plus de numéros situés entre 0 et 14 999, cet intervalle est réservé à 4D. Pour vos propres ressources, utilisez les numéros situés entre 15 000 et 32 767. Rappelez-vous que dès qu'un fichier de ressources est ouvert, il devient le premier maillon de la chaîne des fichiers de ressources, et c'est dans ce fichier que les ressources seront recherchées en premier lieu. En conséquence, si vous stockez dans ce fichier des ressources dont les numéros appartiennent aux intervalles réservés au Système ou à 4D, ces ressources seront utilisées non seulement par les commandes telles que **LIRE RESSOURCE** mais également par les routines internes de l'application 4D elle-même. Si vous n'êtes pas absolument certain de ce que vous faites, n'utilisez pas les intervalles réservés, cela peut conduire à des erreurs système.
- Un fichier de ressources est très structuré et ne peut contenir plus 2 700 ressources. Si vous travaillez avec des fichiers comportant un grand nombre de ressources, il est conseillé de tester ce nombre avant d'ajouter de nouvelles ressources à un fichier (reportez-vous à l'exemple ***Nombre de ressources*** dans la description de la commande **LISTE TYPES RESSOURCE**).

Une fois que vous avez ouvert un fichier de ressources, vous pouvez analyser son contenu à l'aide des commandes **LISTE TYPES RESSOURCE** et **LISTE RESSOURCES**.

Exemple 1

Dans l'exemple suivant, nous cherchons à ouvrir sous Windows le fichier de ressources "MesPrefs.res" situé dans le dossier de la base :

```
$vhResFile:=Ouvrir fichier ressources("MesPrefs";"res ")
```

Sous Mac OS, l'exemple recherchera le fichier “MesPrefs”.

Exemple 2

Cet exemple tente d'ouvrir sous Windows le fichier de ressources "MesPrefs.rsr" situé dans le dossier de la base :

```
$vhResFile:=Ouvrir fichier ressources("MesPrefs";"rsr")
```

Sous Mac OS, l'exemple recherchera le fichier “MesPrefs”.

Exemple 3

L'exemple suivant fait apparaître la boîte de dialogue standard d'ouverture de fichiers, dans laquelle tous les types de documents sont affichés :

```
$vhResFile:=Ouvrir fichier ressources("")
```

Exemple 4

L'exemple suivant fait apparaître la boîte de dialogue standard d'ouverture de fichiers, dans laquelle seuls les documents créés à l'aide de la fonction **_o_Créer fichier ressources** et utilisant le type par défaut sont affichés :

```
$vhResFile:=Ouvrir fichier ressources("";"res ")
Si(OK=1)
   ALERTE("Vous venez d'ouvrir "+Document+.")
   FERMER FICHIER RESSOURCES($vhResFile)
Fin de si
```

Variables et ensembles système

Si le fichier de ressources est correctement ouvert, la variable système OK prend la valeur 1. Si le fichier de ressources n'a pas pu être ouvert ou si l'utilisateur a cliqué sur le bouton Annuler dans la boîte de dialogue standard d'ouverture de fichiers, la variable OK prend la valeur 0 (zéro).

Si le fichier de ressources est correctement ouvert par l'intermédiaire de la boîte de dialogue standard d'ouverture de fichiers, la variable système Document contient le chemin d'accès au fichier.

Gestion des erreurs

Si le fichier de ressources n'a pas pu être ouvert à la suite d'un problème de ressource ou d'E/S, une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

_o_Créer fichier ressources

_o_Créer fichier ressources (resNomFichier {; typeFichier {; *}}) -> Résultat

Paramètre	Type		Description
resNomFichier	Chaîne	⇒	Nom ou chemin d'accès complet du fichier de ressources ou chaîne vide pour afficher la boîte de dialogue standard d'enregistrement de fichiers
typeFichier	Chaîne	⇒	Type de fichier Mac OS (chaîne de 4 caractères), ou extension de fichier Windows (chaîne de 1 à 3 caractères), ou fichier de ressources ("res " / .RES) si omis
*		⇒	Si passé = utiliser la data fork
Résultat	RefDoc	⇐	Numéro de référence du fichier de ressources

Description

Note de compatibilité : Les commandes permettant d'écrire dans des fichiers de ressources sont obsolètes ne doivent plus être utilisées.

_o_ÉCRIRE NOM RESSOURCE

_o_ÉCRIRE NOM RESSOURCE (resType ; resNum ; resName {; resFichier})

Paramètre	Type		Description
resType	Chaîne	⇒	Type de ressource (4 caractères)
resNum	Entier long	⇒	Numéro de référence de ressource (ID)
resName	Chaîne	⇒	Nouveau nom de la ressource
resFichier	RefDoc	⇒	Numéro de référence du fichier de ressource ou Tous les fichiers de ressources ouverts si omis

Description

Note de compatibilité : A compter de 4D v13, les commandes permettant d'écrire dans des fichiers de ressources ne fonctionnent plus et ne doivent plus être utilisées.

_o_ÉCRIRE PROPRIÉTÉS RESSOURCE

_o_ÉCRIRE PROPRIÉTÉS RESSOURCE (resType ; resNum ; resAttr {; resFichier})

Paramètre	Type		Description
resType	Chaîne	⇒	Type de ressource (4 caractères)
resNum	Entier long	⇒	Numéro de référence de ressource (ID)
resAttr	Entier long	⇒	Nouveaux attributs de la ressource
resFichier	RefDoc	⇒	Numéro de référence du fichier de ressource ou Tous les fichiers de ressources ouverts si omis

Description

Note de compatibilité : A compter de 4D v13, les commandes permettant d'écrire dans des fichiers de ressources ne fonctionnent plus et ne doivent plus être utilisées

_o_ÉCRIRE RESSOURCE

_o_ÉCRIRE RESSOURCE (resType ; resNum ; resDonnées {; resFichier})

Paramètre	Type		Description
resType	Chaîne	⇒	Type de ressource (4 caractères)
resNum	Entier long	⇒	Numéro de ressource
resDonnées	BLOB	⇒	Nouveau contenu de la ressource
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou Fichier de ressources courant si omis

Description

Note de compatibilité : A compter de 4D v13, les commandes permettant d'écrire dans des fichiers de ressources ne fonctionnent plus et ne doivent plus être utilisées.

_o_ÉCRIRE RESSOURCE CHAINE

_o_ÉCRIRE RESSOURCE CHAINE (resNum ; resDonnées {; resFichier})

Paramètre	Type		Description
resNum	Entier long	⇒	Numéro de ressource
resDonnées	Chaîne	⇒	Nouveau contenu de la ressource STR
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Commande désactivée

Note de compatibilité : A compter de 4D v13, les commandes permettant d'écrire dans des fichiers de ressources ne fonctionnent plus et ne doivent plus être utilisées.

_o_ÉCRIRE RESSOURCE IMAGE

_o_ÉCRIRE RESSOURCE IMAGE (resNum ; resDonnées {; resFichier})

Paramètre	Type		Description
resNum	Entier long	⇒	Numéro de ressource
resDonnées	Image	⇒	Nouveau contenu de la ressource PICT
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Description

Note de compatibilité : A compter de 4D v13, les commandes permettant d'écrire dans des fichiers de ressources ne fonctionnent plus et ne doivent plus être utilisées

_o_ÉCRIRE RESSOURCE TEXTE

_o_ÉCRIRE RESSOURCE TEXTE (resNum ; resDonnées {; resFichier})

Paramètre	Type		Description
resNum	Entier long	⇒	Numéro de ressource
resDonnées	Chaîne	⇒	Nouveau contenu de la ressource TEXT
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Description

Note de compatibilité : A compter de 4D v13, les commandes permettant d'écrire dans des fichiers de ressources ne fonctionnent plus et ne doivent plus être utilisées

_o_Lire ID ressource composant

_o_Lire ID ressource composant (nomComp ; resType ; resNumOriginal) -> Résultat

Paramètre	Type		Description
nomComp	Chaîne	⇒	Nom du composant référençant la ressource
resType	Chaîne	⇒	Type de ressource (4 caractères), PICT ou STR#
resNumOriginal	Entier long	⇒	Numéro original de la ressource, avant installation du composant
Résultat	Entier long	⇐	Numéro courant de la ressource

Description

Note de compatibilité : Cette commande fonctionnait avec les composants d'ancienne génération, incompatibles avec 4D version 11 et suivantes. Elle est désormais sans effet et ne doit plus être utilisée.

_o_SUPPRIMER RESSOURCE

_o_SUPPRIMER RESSOURCE (resType ; resNum {; resFichier})

Paramètre	Type		Description
resType	Chaîne	⇒	Type de ressource (4 caractères)
resNum	Entier long	⇒	Numéro de ressource
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Note de compatibilité

Depuis 4D v13, les commandes permettant d'écrire dans des fichiers de ressources ne fonctionnent plus et ne doivent plus être utilisées.

_o_TABLEAU VERS LISTE DE CHAÎNES

_o_TABLEAU VERS LISTE DE CHAÎNES (tabChaînes ; resNum {; resFichier})

Paramètre	Type		Description
tabChaînes	Tableau chaîne	⇒	Tableau alpha ou texte (nouveau contenu de la ressource STR#)
resNum	Entier long	⇒	Numéro de ressource
resFichier	RefDoc	⇒	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Description

Note de compatibilité : A compter de 4D v13, les commandes permettant d'écrire dans des fichiers de ressources ne fonctionnent plus et ne doivent plus être utilisées.

Saisie

- AJOUTER ENREGISTREMENT
- Ancien
- DIALOGUE
- Modifie
- MODIFIER ENREGISTREMENT
- NE PAS VALIDER
- REFUSER
- VALIDER
- *_o_AJOUTER SOUS ENREGISTREMENT*
- *_o_MODIFIER SOUS ENREGISTREMENT*

AJOUTER ENREGISTREMENT

AJOUTER ENREGISTREMENT ({laTable}{;}{*})

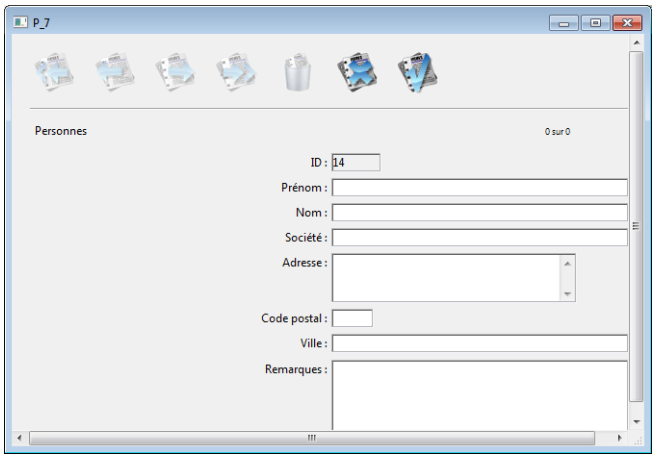
Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle ajouter des données ou Table par défaut si ce paramètre est omis
*		⇒	Cacher les barres de défilement

Description

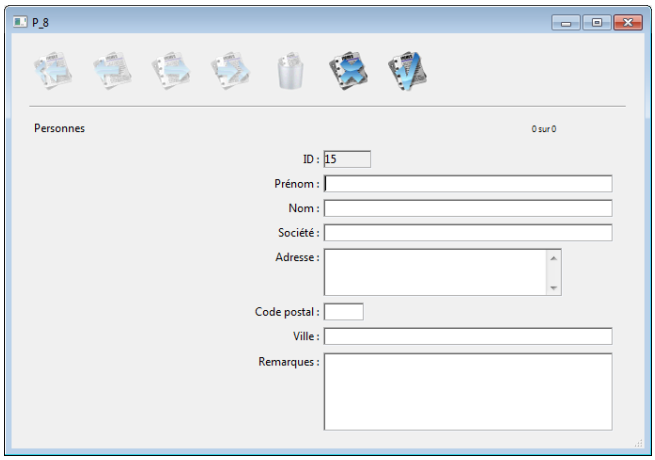
La commande **AJOUTER ENREGISTREMENT** permet à l'utilisateur de créer un nouvel enregistrement dans *laTable* ou dans la table par défaut si ce paramètre est omis.

AJOUTER ENREGISTREMENT crée un nouvel enregistrement pour *laTable*, en fait l'enregistrement courant pour le process courant et l'affiche dans le formulaire entrée courant. En mode Application, une fois que l'utilisateur a validé le nouvel enregistrement, la sélection courante est réduite à ce seul enregistrement.

L'écran suivant présente un formulaire typiquement utilisé pour la saisie de données :



Le formulaire est affiché dans la fenêtre se trouvant au premier plan du process. Elle comporte des barres de défilement et une case de contrôle de taille. Si vous passez le paramètre optionnel astérisque (*), les barres de défilement n'apparaissent pas et la fenêtre du formulaire ne peut être réduite :



AJOUTER ENREGISTREMENT affiche le formulaire jusqu'à ce que l'utilisateur valide ou annule l'enregistrement. Si l'utilisateur ajoute plusieurs enregistrements, la commande doit être appelée pour chaque nouvel enregistrement.

L'enregistrement est sauvegardé si l'utilisateur clique sur un bouton du type Valider ou appuie sur la touche Entrée, ou encore si la commande **VALIDER** est exécutée.

L'enregistrement n'est pas sauvegardé si l'utilisateur clique sur un bouton du type **Annuler** ou appuie sur la touche d'annulation (**Echap** sous Windows, **Esc** sous Mac OS), ou encore si la commande **NE PAS VALIDER** est exécutée.

Note : Cette commande ne nécessite pas que *laTable* soit en mode lecture/écriture. Elle peut être utilisée même lorsque la table est en mode lecture seulement (cf. section **Verrouillage d'enregistrements**).

Après un appel à **AJOUTER ENREGISTREMENT**, la variable système OK prend la valeur 1 si l'enregistrement est validé et 0 s'il est annulé.

Note : Même lorsqu'il est annulé, l'enregistrement reste en mémoire et peut être sauvegardé avec la commande **STOCKER ENREGISTREMENT** si celle-ci est exécutée avant que le pointeur d'enregistrement courant ne soit modifié.

Exemple 1

L'exemple suivant est une boucle souvent utilisée pour créer des enregistrements dans une base :

```
FORM FIXER ENTREE([Clients];"SaisieClients") ` Désigner le formulaire entrée de la table [Clients]
Repeter ` Boucle jusqu'à ce que l'utilisateur annule
   AJOUTER ENREGISTREMENT([Clients];*) ` Ajouter un enregistrement dans la table [Clients]
Jusque(OK=0) ` Jusqu'à ce que l'utilisateur annule
```

Exemple 2

L'exemple suivant permet de rechercher un client dans la base. Le déroulement de la méthode dépend du résultat de la recherche. Si aucun client n'a été trouvé, l'utilisateur est autorisé à créer un nouveau client à l'aide de la commande **AJOUTER ENREGISTREMENT**. Si au moins un client a été trouvé, le premier enregistrement est affiché pour modification, à l'aide de la commande **MODIFIER ENREGISTREMENT** :

```
LECTURE ECRITURE([Clients])
FORM FIXER ENTREE([Clients];"Entrée1") ` Désigner le formulaire entrée
vlClientNo:=Num(Demander("Saisissez un numéro de client :") ` On récupère le numéro du client
Si(OK=1)
   CHERCHER([Clients];[Clients]ClientNo=vlClientNo) ` Recherche du client
   Si(Enregistrements trouves([Clients])=0) ` Si aucun client n'a été trouvé…
      AJOUTER ENREGISTREMENT([Clients]) `Ajout d'un nouveau client
   Sinon
      Si(Non(Enregistrement verrouille([Clients])))
         MODIFIER ENREGISTREMENT([Clients]) `Modifier l'enregistrement
         LIBERER ENREGISTREMENT([Clients])
      Sinon
         ALERTE("Cet enregistrement est en train d'être modifié.")
      Fin de si
   Fin de si
Fin de si
```

Variables et ensembles système

La variable système OK prend la valeur 1 si l'enregistrement est validé et 0 s'il est annulé.

Ancien

Ancien (leChamp) -> Résultat

Paramètre	Type		Description
leChamp	Champ	➡	Champ dont vous voulez obtenir l'ancienne valeur
Résultat	Expression	⮌	Valeur originale de champ

Description

La commande **Ancien** retourne la valeur qui était stockée dans *leChamp* avant qu'il n'ait été modifié par programmation ou pendant la saisie de données.

A chaque fois que vous changez d'enregistrement courant pour une table, 4D crée et maintient en mémoire un double de l'"image" du nouvel enregistrement courant au moment où il est chargé. Lorsque vous modifiez un enregistrement, vous travaillez avec l'image réelle de l'enregistrement, et non son double. Ce double est effacé lorsque que vous changez à nouveau d'enregistrement courant.

Ancien retourne la valeur de *leChamp* telle qu'elle est stockée dans le double de l'enregistrement. Autrement dit, pour un enregistrement existant, **Ancien** retourne la valeur du champ telle qu'elle avait été sauvegardée sur disque. Pour un enregistrement qui vient d'être créé, **Ancien** retourne la valeur vide par défaut correspondant au type de *leChamp*. Par exemple, si *leChamp* est de type Alpha, **Ancien** retourne une chaîne vide. Si champ est de type numérique, **Ancien** retourne zéro (0), etc.

Ancien fonctionne avec *leChamp* de la même manière, que le champ ait été modifié par programmation ou suite à des modifications effectuées par un utilisateur.

La fonction accepte tous les types de champs.

Pour restaurer la valeur originale d'un champ, assignez-lui la valeur retournée par **Ancien**.

Note : Pour des raisons techniques, dans le cas des champs de type Image et BLOB, l'expression retournée par **Ancien** ne peut pas être directement utilisée comme paramètre d'une autre commande. Il est nécessaire de faire transiter la valeur par une variable intermédiaire. Par exemple :

```
 `Ne PAS écrire (provoque une erreur de syntaxe) :
$taille :=Taille BLOB(Ancien([LaTable]LeBlob)) `INCORRECT

 `Ecrire :
$ancienBlob :=Ancien([LaTable]LeBlob)
$taille :=Taille BLOB($ancienBlob) `CORRECT
```

DIALOGUE

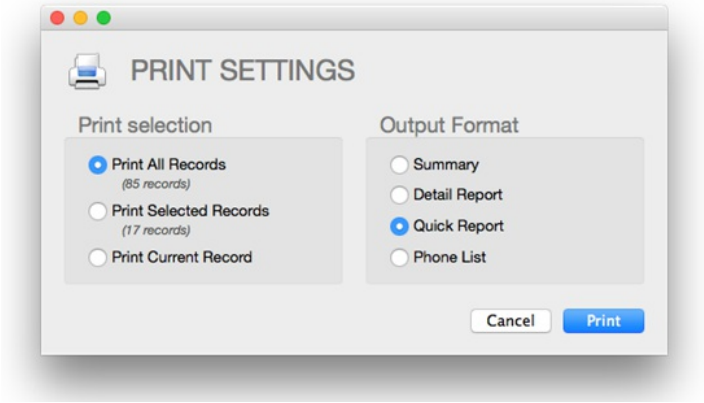
DIALOGUE ({laTable ;} formulaire {; *})

Paramètre	Type		Description
laTable	Table	⇒	Table à laquelle appartient le formulaire Si omis : Table par défaut ou utilisation d'un formulaire projet
formulaire	Chaîne	⇒	Nom du formulaire table ou projet à afficher comme dialogue
*	Opérateur	⇒	Utiliser le même process

Description

La commande **DIALOGUE** présente le *formulaire* à l'utilisateur. Cette commande est souvent utilisée pour récupérer dans des variables des données fournies par l'utilisateur, ou pour lui présenter différentes informations, comme un choix d'options pour effectuer une opération.

Il est courant d'afficher le *formulaire* dans une fenêtre modale créée à l'aide de la commande **Creer fenetre**.
Voici un exemple typique de boîte de dialogue pouvant être affichée avec la commande **DIALOGUE** :



Utilisez **DIALOGUE** plutôt que **ALERTE**, **CONFIRMER** ou **Demander** lorsque les informations à afficher ou à recueillir sont plus complexes que celles que peuvent gérer ces trois autres commandes.

Note : Dans les bases de données converties, il est possible d'interdire la saisie dans les champs dans les boîtes de dialogue (et donc de limiter la saisie aux seules variables) via une option des Préférences de 4D (page Compatibilité). Cette restriction correspond au fonctionnement des anciennes versions de 4D.

A la différence d'**AJOUTER ENREGISTREMENT** et de **MODIFIER ENREGISTREMENT**, **DIALOGUE** n'utilise pas le formulaire entrée courant. Vous devez spécifier, dans le paramètre *formulaire*, le formulaire (formulaire projet ou formulaire table) à utiliser. De même, aucun ensemble de boutons n'est placé par défaut s'ils sont omis dans le formulaire. Dans ce cas, seule la touche **Echap** (Windows) ou **Esc** (Mac OS) permet de quitter le formulaire.

Le dialogue est validé si l'utilisateur clique sur le bouton de validation ou appuie sur la touche **Entrée**, ou si la commande **VALIDER** est exécutée.
A noter que la validation n'entraîne pas la sauvegarde : si le dialogue comporte des champs, vous devez appeler explicitement la commande **STOCKER ENREGISTREMENT** pour stocker les données éventuellement modifiées.

Le dialogue est annulé si l'utilisateur clique sur le bouton d'annulation, appuie sur la touche d'annulation (**Echap** sous Windows, **Esc** sous Mac OS), ou si la commande **NE PAS VALIDER** est exécutée.

Si vous passez le paramètre facultatif * , le formulaire est chargé et affiché dans la dernière fenêtre ouverte du process courant et la commande termine son exécution en laissant le formulaire actif à l'écran.
Ce formulaire réagit alors "normalement" aux actions de l'utilisateur et est fermé via une action standard ou lorsque du code 4D lié au formulaire (méthode objet ou méthode formulaire) appelle la commande **NE PAS VALIDER** ou **VALIDER**. Si le process courant se termine, les formulaires créés de cette façon sont automatiquement fermés en simulant un **NE PAS VALIDER**. Ce mode d'ouverture est particulièrement utile pour afficher une palette flottante en rapport avec un document, sans pour autant nécessiter un autre process.

Notes :

- Vous devez créer une fenêtre avant d'appeler l'instruction **DIALOGUE(form;*)**, il n'est pas possible d'utiliser la fenêtre du dialogue en cours dans le process ni la fenêtre créée par défaut pour chaque process. Dans le cas contraire, l'erreur -9909 est générée.
- Lorsque le paramètre * est utilisé, la fenêtre est refermée automatiquement à la suite d'une action standard ou de l'appel de la commande **NE PAS VALIDER** ou **VALIDER**. Vous ne devez pas gérer vous-même la fermeture de la fenêtre.

Exemple 1

L'exemple suivant illustre l'utilisation de la commande **DIALOGUE** pour spécifier des critères de recherche. Un formulaire personnalisé contenant les variables *vNom* et *vPays* permet à l'utilisateur de saisir ses critères :

```
Creer fenetre(10;40;370;220) ` Créer une fenêtre modale
DIALOGUE("Form Recherche") ` Afficher le dialogue de recherche
FERMER FENETRE ` Nous n'avons plus besoin de la fenêtre
Si(OK=1) ` Si le dialogue est validé
   CHERCHER([Société];[Société]Nom=vNom;*)
   CHERCHER( & [Société]Payst =vPays)
Fin de si
```

Exemple 2

L'exemple suivant permet de créer une palette d'outils :

```
 `Affichage palette d'outils
$window_palette:=Creer fenetre formulaire("tools";Form fenêtre palette)
DIALOGUE("tools";*) `Rend la main immédiatement
 `Affichage fenêtre document principal
$window_document:=Creer fenetre formulaire("doc";Form fenêtre standard)
DIALOGUE("doc")
```

Variables et ensembles système

Si l'utilisateur valide le dialogue, la variable système OK prend la valeur 1, si le dialogue est annulé OK prend la valeur 0.

Modifie

Modifie (leChamp) -> Résultat

Paramètre	Type		Description
leChamp	Champ	➡	Champ dont vous voulez tester la modification
Résultat	Booléen	⮌	Vrai si une nouvelle valeur a été assignée au champ, sinon Faux

Description

Modifie retourne **Vrai** si une valeur a été assignée par programmation au champ *leChamp* ou s'il a été modifié lors de la saisie de données. La commande **Modifie** ne fonctionne que lorsqu'elle est appelée dans le cadre d'une méthode formulaire (ou d'une sous-méthode appelée par la méthode formulaire).

Attention, cette commande ne retourne une valeur significative qu'à l'intérieur d'un même cycle d'exécution. Elle est notamment remise à **Faux** pour tous les événements formulaires correspondant à l'ancien cycle d'exécution **_o_Pendant** (Sur clic, Sur après frappe clavier...).

Dans le cas de la saisie de données, un champ est considéré comme modifié à partir du moment où un utilisateur l'édite (et change ou non sa valeur originale) puis le quitte pour un autre champ ou pour cliquer sur un objet de formulaire. Notez que le fait qu'un utilisateur active puis quitte un champ à l'aide de la touche Tabulation ne suffit pas en soi à ce que **Modifie** retourne **Vrai**. Le champ doit avoir été réellement modifié pour que **Modifie** retourne **Vrai**.

Dans le cas de l'exécution d'une méthode, un champ est considéré comme modifié si une valeur lui a été assignée (différente ou non de sa valeur précédente).

Note : La commande **Modifie** retourne toujours **Vrai** après l'exécution des commandes **EMPILER ENREGISTREMENT** et **DEPILER ENREGISTREMENT**.

Dans tous les cas, pour savoir si la valeur d'un champ a été effectivement modifiée, utilisez la commande **Ancien**.

Note : Bien que la fonction **Modifie** puisse être appliquée à tout type de champ, si vous l'utilisez conjointement avec la fonction **Ancien**, vous devez dans ce cas tenir compte des restrictions liées à cette fonction. Reportez-vous à la description de la commande **Ancien**.

Pendant la saisie de données, il est généralement plus pratique d'effectuer des opérations dans des méthodes objet à l'aide de la commande **Evenement formulaire** que d'utiliser la fonction **Modifie** dans des méthodes formulaires. Comme une méthode objet reçoit l'événement Sur données modifiées à chaque fois qu'un champ est modifié, utiliser une telle méthode équivaut à appeler **Modifie** dans une méthode formulaire.

Exemple 1

L'exemple suivant teste si le champ *[Commandes]Quantité* ou le champ *[Commandes]Prix* a été modifié. Si c'est le cas, le champ *[Commandes]Total* est recalculé :

```
Si((Modifie([Commandes]Quantité)|(Modifie([Commandes]Prix))
   [Commandes]Total:=[Commandes]Quantité*[Commandes]Prix
Fin de si
```

Notez que le même résultat aurait pu être obtenu en utilisant la seconde ligne de cette méthode en tant que méthode objet des champs *[Commandes]Quantité* et *[Commandes]Prix* dans le cadre de l'événement formulaire Sur données modifiées.

Exemple 2

Vous sélectionnez un enregistrement de la table *[uneTable]*, puis vous appelez plusieurs sous-routines qui sont susceptibles de modifier le champ *[uneTable]Champ important* mais sans provoquer de sauvegarde de l'enregistrement. A la fin de la méthode principale, vous pouvez utiliser la commande **Modifie** pour déterminer si vous devez stocker l'enregistrement :

```
  ` L'enregistrement a été sélectionné comme enregistrement courant
  ` Puis vous effectuez des actions à l'aide des sous-routines
FAIRE QUELQUE CHOSE
FAIRE AUTRE CHOSE
NE PAS OUBLIER DE FAIRE CA
  ` ...
  ` Enfin, vous testez le champ pour déterminer s'il faut stocker l'enregistrement
Si(Modifie([uneTable]Champ important))
   STOCKER ENREGISTREMENT([uneTable])
Fin de si
```

MODIFIER ENREGISTREMENT

MODIFIER ENREGISTREMENT ({laTable}{;}{*})

Paramètre	Type		Description
laTable	Table	⇨	Table dans laquelle modifier des données ou Table par défaut si ce paramètre est omis
*		⇨	Cacher les barres de défilement

Description

La commande **MODIFIER ENREGISTREMENT** permet à l'utilisateur de modifier l'enregistrement courant de *laTable*, ou de la table par défaut si ce paramètre est omis. **MODIFIER ENREGISTREMENT** charge depuis le disque l'enregistrement courant pour le process en cours (s'il n'est pas déjà chargé par un autre utilisateur/process) et l'affiche dans le formulaire entrée courant. S'il n'y a pas d'enregistrement courant, **MODIFIER ENREGISTREMENT** ne fait rien. **MODIFIER ENREGISTREMENT** ne change pas la sélection courante.

Le formulaire est affiché dans la fenêtre se trouvant au premier plan du process. Elle comporte des barres de défilement et une case de contrôle de taille. Si vous passez le paramètre optionnel astérisque (*), ces éléments n'apparaîtront pas.

Pour que vous puissiez utiliser **MODIFIER ENREGISTREMENT**, l'enregistrement courant doit être en Lecture/écriture et ne doit pas être verrouillé.

Si le formulaire comporte des boutons de navigation parmi les enregistrements de la sélection, ils restent utilisables, ce qui permet à l'utilisateur de modifier des enregistrements puis de se déplacer pour en modifier d'autres.

L'enregistrement est sauvegardé si l'utilisateur clique sur un bouton du type Valider ou appuie sur la touche **Entrée**, ou encore si la commande **VALIDER** est exécutée.

L'enregistrement n'est pas sauvegardé si l'utilisateur clique sur un bouton du type **Annuler** ou appuie sur la touche d'annulation (**Echap** sous Windows, **Esc** sous Mac OS), ou encore si la commande **NE PAS VALIDER** est exécutée.

Après un appel à **MODIFIER ENREGISTREMENT**, la variable système OK prend la valeur 1 si l'enregistrement est validé, et 0 lorsqu'il est annulé.

Note : Même lorsqu'il est annulé, l'enregistrement reste en mémoire et peut être sauvegardé par la commande **STOCKER ENREGISTREMENT** si celle-ci est appelée avant que le pointeur d'enregistrement courant ne soit modifié.

Dans le cadre d'un **MODIFIER ENREGISTREMENT**, si l'utilisateur n'effectue aucune modification dans l'enregistrement et le valide, l'enregistrement ne sera pas considéré comme modifié et ne sera pas sauvegardé une nouvelle fois. Les actions telles que le changement de la valeur d'une variable, la sélection de cases à cocher ou de boutons radio ne sont pas qualifiées de modifications. Seule la modification de la valeur d'un champ, par le biais d'une saisie manuelle ou d'une méthode, provoque une nouvelle sauvegarde de l'enregistrement.

Exemple

Reportez-vous au second exemple de la commande **AJOUTER ENREGISTREMENT**.

Variables et ensembles système

La variable système OK prend la valeur 1 lorsque l'enregistrement est validé et 0 lorsqu'il est annulé. OK ne prend une valeur qu'après que l'enregistrement ait été effectivement validé ou annulé.

NE PAS VALIDER

NE PAS VALIDER
Ne requiert pas de paramètre

Description

La commande **NE PAS VALIDER** doit être employée dans une méthode objet ou formulaire (ou une sous-routine) pour :

- annuler la création ou la modification d'un enregistrement ou un sous-enregistrement — dont les données ont été saisies à la suite d'un **AJOUTER ENREGISTREMENT**, **MODIFIER ENREGISTREMENT**, **_o_AJOUTER SOUS ENREGISTREMENT** ou **_o_MODIFIER SOUS ENREGISTREMENT**.
- annuler un formulaire affiché par l'intermédiaire de la commande **DIALOGUE**.
- quitter un formulaire affichant une sélection d'enregistrements — à l'aide de **VISUALISER SELECTION** ou **MODIFIER SELECTION**.
- annuler l'impression d'une ligne sur le point d'être imprimée à l'aide de la commande **Imprimer ligne** (voir ci-dessous).

Dans le contexte de la saisie, **NE PAS VALIDER** effectue la même action que lorsque l'utilisateur utilise la touche d'annulation (**Esc**).
NE PAS VALIDER est fréquemment exécutée à la suite de la sélection d'une commande de menu. **NE PAS VALIDER** est également souvent appelée dans la méthode objet d'un bouton auquel la propriété "Pas d'action" a été associée.

Cette commande peut également être placée dans la méthode de la case de fermeture (optionnelle) d'une fenêtre créée par la commande **Creer fenetre**. Si la fenêtre comporte une case de menu Système, **NE PAS VALIDER** et **VALIDER** peuvent être appelées dans la méthode à exécuter lorsque l'utilisateur double-clique sur la case du menu Système ou sélectionne la commande de menu **Fermeture**.
Il n'est pas possible d'enchaîner plusieurs **NE PAS VALIDER**. En d'autres termes, l'exécution consécutive de deux commandes **NE PAS VALIDER** dans une méthode en réponse à un événement aura le même résultat que l'exécution d'une seule.

Enfin, cette commande peut être utilisée dans l'événement formulaire Sur impression corps, dans le cadre de l'utilisation de la commande **Imprimer ligne**. Dans ce contexte, la commande **NE PAS VALIDER** suspend l'impression de la ligne sur le point d'être imprimée, puis la reprend page suivante. Ce mécanisme permet de gérer le manque de place ou les sauts de page lors des impressions des lignes.

Note : Ce fonctionnement est différent de celui de l'instruction **SAUT DE PAGE(*)** qui provoque l'annulation de TOUTES les lignes en attente d'impression.

Exemple

Reportez-vous à l'exemple de la commande **FIXER TAQUET IMPRESSION**.

Variables et ensembles système

Lorsque la commande **NE PAS VALIDER** est exécutée (formulaire annulé ou annulation d'impression), la variable système OK prend la valeur 0.

REFUSER

REFUSER {(leChamp)}

Paramètre	Type		Description
leChamp	Champ	⇨	Champ dont la saisie doit être refusée

Description

REFUSER accepte deux syntaxes. Dans la première syntaxe, **REFUSER** n'a pas de paramètre. Dans ce cas, la commande rejette la totalité de la saisie et force l'utilisateur à rester dans le formulaire. La seconde syntaxe permet de ne refuser que *leChamp* et force l'utilisateur à rester dans le champ.

Note : Nous vous conseillons d'utiliser en priorité les outils intégrés de validation de saisie de 4D, avant de faire appel à cette commande.

La première syntaxe de **REFUSER** est utilisée pour empêcher l'utilisateur de valider un enregistrement incomplet. Vous pouvez parvenir au même résultat sans utiliser **REFUSER** : associez la touche **Entrée** à un bouton n'effectuant "Pas d'action" et utilisez les commandes **VALIDER** et **NE PAS VALIDER** pour valider ou annuler l'enregistrement, une fois que les champs ont été correctement remplis. Il est recommandé d'employer cette seconde technique plutôt que d'utiliser la première syntaxe de **REFUSER**.

En général, vous employez la première syntaxe de **REFUSER** pour empêcher l'utilisateur de valider un enregistrement incomplet ou comportant des valeurs incorrectes. Si l'utilisateur tente de valider l'enregistrement, l'exécution de **REFUSER** provoque l'annulation de cette commande et l'enregistrement reste affiché dans le formulaire. L'utilisateur doit alors recommencer la saisie jusqu'à ce que les valeurs soient considérées comme correctes ou annuler l'enregistrement.

Le meilleur emplacement pour la commande **REFUSER**, lorsque vous utilisez cette syntaxe, est la méthode objet d'un bouton de type Valider associé à la touche de validation. De cette manière, la validation n'est possible que lorsque l'enregistrement est accepté, et l'utilisateur ne peut pas "forcer" la validation en appuyant sur la touche **Entrée**.

La seconde syntaxe de **REFUSER** utilise le paramètre *leChamp*. Dans ce cas, le curseur reste dans la zone de saisie du champ, ce qui oblige l'utilisateur à saisir une valeur correcte.
Avec cette syntaxe, la commande **REFUSER** doit impérativement être appelée dans l'événement formulaire Sur données modifiées.Vous devez placer cette syntaxe de **REFUSER** soit dans la méthode formulaire, soit dans la méthode objet de la zone de saisie. Si vous utilisez **REFUSER** avec le formulaire "pleine page" d'un sous-formulaire, placez-la dans la méthode formulaire ou une méthode objet du formulaire "pleine page". Lorsqu'elle est utilisée avec des champs de sous-formulaires, cette commande ne fait rien.

Vous pouvez utiliser la commande **SELECTIONNER TEXTE** pour sélectionner, à l'intérieur du champ, les valeurs qui ont été refusées.

Exemple 1

L'exemple suivant illustre la première syntaxe de **REFUSER**, placée dans la méthode objet d'un bouton Valider. La touche **Entrée** a été définie comme équivalent clavier pour ce bouton. Cela signifie que même si l'utilisateur appuie sur cette touche pour valider l'enregistrement, la méthode objet du bouton sera exécutée. L'enregistrement est une transaction bancaire. Si la transaction est un chèque, un numéro de chèque doit être saisi. S'il n'y a pas de numéro, la validation est refusée :

```
Au cas ou
   :(([Opération]Trans="Chèque") & ([Opération]Numéro="")) ` Si c'est un chèque sans numéro...
      ALERTE("Veuillez saisir le numéro du chèque.") ` Alerter l'utilisateur
      REFUSER ` Refuser la saisie
      ALLER A OBJET([Opération]Numéro) ` Placer le curseur dans le champ "numéro de chèque"
Fin de cas
```

Exemple 2

L'exemple suivant est une partie de la méthode objet d'un champ *[Employés]Salaire*. La méthode objet teste si la valeur de ce champ est inférieure à 10 000 Euros et la refuse si c'est le cas. Vous pourriez effectuer le même contrôle en spécifiant une valeur minimum pour le champ, dans l'éditeur de formulaires du mode Développement :

```
Au cas ou
   :(Evenement formulaire=Sur données modifiées)
      Si([Employés]Salaire<10000)
         ALERTE("Le salaire annuel doit être supérieur à 10 000 Euros.")
         REFUSER([Employés]Salaire)
      Fin de si
Fin de cas
```

VALIDER

VALIDER
Ne requiert pas de paramètre

Description

La commande **VALIDER** doit être employée dans une méthode objet ou formulaire (ou une sous-routine) pour :

- valider un enregistrement ou un sous-enregistrement créé ou modifié — dont les données ont été saisies à la suite d'un **AJOUTER ENREGISTREMENT**, **MODIFIER ENREGISTREMENT**, **_o_AJOUTER SOUS ENREGISTREMENT** ou **_o_MODIFIER SOUS ENREGISTREMENT**.
- valider un formulaire affiché par l'intermédiaire de la commande **DIALOGUE**.
- quitter un formulaire affichant une sélection d'enregistrements — à l'aide de **VISUALISER SELECTION** ou **MODIFIER SELECTION**.

VALIDER effectue la même action que lorsque l'utilisateur appuie sur la touche **Entrée**. Une fois que le formulaire a été validé, la variable système OK prend la valeur 1.

VALIDER est fréquemment exécutée à la suite de la sélection d'une commande de menu. **VALIDER** est également souvent appelée dans la méthode objet d'un bouton auquel la propriété "Pas d'action" a été associée.

Enfin, cette commande peut être placée dans la méthode de la case de fermeture (optionnelle) d'une fenêtre créée par la commande **Creer fenetre**. Si la fenêtre comporte une case de menu Système, **VALIDER** et **NE PAS VALIDER** peuvent être appelées dans la méthode à exécuter lorsque l'utilisateur double-clique sur la case du menu Système ou sélectionne la commande de menu **Fermeture**.

Il n'est pas possible d'enchaîner plusieurs **VALIDER**. En d'autres termes, l'exécution consécutive de deux commandes **VALIDER** dans une méthode en réponse à un événement aura le même résultat que l'exécution d'une seule.

_o_AJOUTER SOUS ENREGISTREMENT

_o_AJOUTER SOUS ENREGISTREMENT (sousTable ; formulaire {; *})

Paramètre	Type		Description
sousTable	Sous-table	⇒	Sous-table à utiliser pour la saisie des données
formulaire	Chaîne	⇒	Formulaire à utiliser
*		⇒	Masquer les barres de défilement

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_MODIFIER SOUS ENREGISTREMENT

_o_MODIFIER SOUS ENREGISTREMENT (sousTable ; formulaire {; *})

Paramètre	Type		Description
sousTable	Sous-table	⇒	Sous-table à utiliser pour la saisie de données
formulaire	Chaîne	⇒	Formulaire à utiliser pour la saisie
*		⇒	Cacher les barres de défilement

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Sauvegarde

- Méthode base Sur arrêt sauvegarde
- Méthode base Sur démarrage sauvegarde
- Fichier historique
- FICHIER HISTORIQUE VERS JSON
- FIXER FICHIER HISTORIQUE
- INTEGRER FICHIER HISTORIQUE MIROIR
- LIRE INFORMATION RESTITUTION
- LIRE INFORMATION SAUVEGARDE
- Nouveau fichier historique
- RESTITUER
- SAUVEGARDER
- VERIFIER FICHIER HISTORIQUE
- *_o_INTEGRER FICHIER HISTORIQUE*

Méthode base Sur arrêt sauvegarde

La **Méthode base Sur arrêt sauvegarde** est appelée à chaque fois qu'une sauvegarde de la base vient de se terminer. Les causes de l'arrêt de la sauvegarde peuvent être la fin de la copie, l'interruption par l'utilisateur ou une erreur.
Tous les environnements 4D sont concernés : 4D (tous modes), 4D Server ainsi que les applications 4D compilées et fusionnées avec 4D Volume Desktop.

La **Méthode base Sur arrêt sauvegarde** permet de vérifier que la sauvegarde s'est correctement déroulée. Elle reçoit dans le paramètre *$1* une valeur indiquant le statut de la sauvegarde à l'issue de son exécution :

- Si la sauvegarde s'est terminée normalement, *$1* vaut 0.
- Si la sauvegarde a été interrompue à la suite d'une erreur ou par l'utilisateur, *$1* est différent de 0.
  - Si la sauvegarde a été stoppée par la **Méthode base Sur démarrage sauvegarde** (*$0* # 0), *$1* retourne le code effectivement retourné dans le paramètre *$0*. Ce principe vous permet de mettre en place un système de gestion d'erreurs personnalisé.
  - Si la sauvegarde a été stoppée à la suite d'une erreur, le code de l'erreur est retourné dans *$1*.

  Dans tous les cas, vous pouvez obtenir des informations sur l'erreur à l'aide de la commande **LIRE INFORMATION SAUVEGARDE**.

Note : Vous devez impérativement déclarer le paramètre *$1* (entier long) dans la méthode base :

```
C_ENTIER LONG($1)
```

Il est important de noter qu'en cas d'erreur durant la sauvegarde (disque plein, support inaccessible...), les informations relatives à l'erreur sont uniquement affichées dans le moniteur de 4D Server ou dans le CSM, et reportées dans le journal des sauvegardes. Aucune boîte de dialogue d'alerte n'est affichée et la variable *error* n'est pas modifiée. Si vous souhaitez pouvoir notifier l'administrateur qu'une erreur s'est produite, en particulier dans le contexte d'une application en mode client/serveur, il est nécessaire d'utiliser la **Méthode base Sur arrêt sauvegarde**.

Méthode base Sur démarrage sauvegarde

La **Méthode base Sur démarrage sauvegarde** est appelée à chaque fois qu'une sauvegarde de la base est sur le point d'avoir lieu (sauvegarde manuelle, sauvegarde automatique périodique ou via la commande **SAUVEGARDER**).
Tous les environnements 4D sont concernés : 4D (tous modes), 4D Server ainsi que les applications 4D compilées et fusionnées avec 4D Volume Desktop.

La **Méthode base Sur démarrage sauvegarde** permet de contrôler le déclenchement de la sauvegarde. Au sein de la méthode, vous devez retourner dans le paramètre $0 une valeur autorisant ou refusant la sauvegarde :

- si $0 = 0, vous autorisez la sauvegarde.
- si $0 # 0, vous n'autorisez pas la sauvegarde. L'opération est annulée et une erreur est retournée. Vous pouvez récupérer l'erreur à l'aide de la commande **LIRE INFORMATION SAUVEGARDE**.

Vous pouvez utiliser cette méthode base pour contrôler les conditions d'exécution de la sauvegarde (utilisateur, date de la dernière sauvegarde, etc.).

Note : Vous devez impérativement déclarer le paramètre *$0* (entier long) dans la méthode base :

```
C_ENTIER LONG($0)
```

Fichier historique

Fichier historique -> Résultat

Paramètre	Type		Description
Résultat	Chaîne	➲	Nom long du fichier d'historique de la base

Description

La commande **Fichier historique** retourne le nom long (c'est-à-dire le chemin d'accès complet au fichier, y compris son nom) du fichier d'historique courant de la base ouverte.

Si la base fonctionne sans fichier d'historique, la fonction retourne une chaîne vide et la variable système OK prend la valeur 0.

Si la base fonctionne avec un fichier d'historique, la variable système OK prend la valeur 1. Le chemin d'accès retourné par la commande est exprimé avec la syntaxe de la plate-forme courante.

ATTENTION : Si vous exécutez cette commande depuis un poste 4D Client, seul le nom du fichier d'historique est retourné, pas le nom long.

Variables et ensembles système

- Si la base fonctionne sans fichier d'historique, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.
- Si le fichier d'historique devient inaccessible au cours de la session de travail, l'erreur 1274 est générée et 4D Server ne permet plus aux utilisateurs d'écrire ou de modifier des données. Lorsque le fichier d'historique est de nouveau accessible, il est nécessaire d'effectuer une sauvegarde.

FICHIER HISTORIQUE VERS JSON

FICHIER HISTORIQUE VERS JSON (cheminDossierDest {; tailleMax {; cheminHistorique {; attribChamp}}})

Paramètre	Type		Description
cheminDossierDest	Texte	⇒	Chemin d'accès du dossier de destination du fichier sauvegardé
tailleMax	Entier long	⇒	Taille maximale du fichier JSON à créer (octets)
cheminHistorique	Texte	⇒	Chemin d'accès du fichier d'historique à exporter ; utiliser l'historique courant si omis
attribChamp	Entier long	⇒	Attribut de description du champ : 1 = utiliser numéro (défaut), 2 = utiliser nom

Description

La commande **FICHIER HISTORIQUE VERS JSON** sauvegarde au format JSON le fichier d'historique courant ou tout fichier d'historique spécifié.

Lorsqu'un fichier d'historique (fichier binaire) est sauvegardé au format JSON, son contenu peut alors être lu et interprété par l'administrateur de la base ou tout utilisateur afin d'analyser les événements de la base, par exemple.

Dans *cheminDossierDest*, passez le chemin du dossier dans lequel vous souhaitez stocker le fichier JSON. Le fichier sera nommé **JournalExport.json**.

Par défaut, la taille maximale du fichier JSON exporté est de 10 Mo. Lorsque cette taille est atteinte, le fichier est refermé et un nouveau fichier est créé. Limiter la taille de chaque fichier JSON réduit la quantité de mémoire requise pour analyser les fichiers. Vous pouvez modifier la taille maximale du fichier exporté en passant une valeur (en octets) dans le paramètre *tailleMax*. Passer 0 rétablit la valeur par défaut (10 Mo). Passer une valeur négative indique une taille illimitée.

Par défaut, si le paramètre *cheminHistorique* est omis, la commande sauvegarde le fichier d'historique courant. Si vous voulez exporter en JSON un fichier d'historique spécifique, passez son chemin d'accès dans le paramètre *cheminHistorique*. Ce fichier d'historique doit avoir l'extension ".journal". Si vous souhaitez exporter un fichier d'historique archivé (extension ".4bl"), vous devez au préalable le convertir à l'aide de la commande **RESTITUER**. Vous pouvez passer une chaîne vide ("") afin d'afficher la boîte de dialogue standard d'ouverture de fichier, permettant à l'utilisateur de sélectionner le fichier d'historique à traiter. Le chemin du fichier sélectionné est retourné dans la variable système **Document**.

Note : Lorsque la commande sauvegarde le fichier d'historique courant, la base n'est pas verrouillée. De nouvelles opérations peuvent être exécutées tandis que le fichier est écrit sur le disque -- ces opérations ne seront pas incluses dans le fichier sauvegardé.

Lorsque vous exportez le fichier d'historique courant, le paramètre *attribChamp* vous permet de définir la manière dont les champs doivent être désignés dans les attributs exportés : via leur numéro (défaut) ou via leur nom. Vous pouvez passer une des constantes suivantes, placées dans le thème de constantes "**Sauvegarde et restitution**" :

Constante	Type	Valeur	Comment
Attribut champ nom	Entier long	2	Les champs sont identifiés par leur nom. Exemple: {"Nom":"Jones"}
Attribut champ numéro	Entier long	1	Les champs sont identifiés par leur numéro (défaut si omis). Exemple: {"5":"Jones"}.

Note : Lorsque vous exportez un fichier d'historique externe, les champs sont toujours désignés par leur numéro.

Le fichier JSON sauvegardé contient toutes les opérations enregistrées dans l'historique sous la forme d'un tableau d'objets JSON. Chaque objet contient plusieurs propriétés décrivant l'opération. Exemple :

```
[
   {
      "operationType":25,
      "operationName":"Modify record",
      "operationNumber":45,
      "contextID":37,
      "timeStamp":"2015-06-11T09:13:17.138Z",
      "dataLen":42,
      "recordNumber":4,
      "tableID":"5AFA15123F991C43B6ACF8B46A914BD0",
      "tableName":"elem",
      "fields": {
            "1": "primkey5",
            "2": -5,
            "5": "data 25"
        },
      "primaryKey": "8"
   },
   {
      "operationType":23,
      "operationName":"Save seqnum",
      "operationNumber":46,
      "contextID":37,
      "timeStamp":"2015-06-11T09:13:17.138Z",
      "sequenceNumber":23,
      "tableID":"5AFA15123F991C43B6ACF8B46A914BD0",
      "tableName":"elem"
   },
   {
      "operationType":24,
      "operationName":"Create record",
      "operationNumber":47,
      "contextID":37,
      "timeStamp":"2015-06-11T09:13:17.138Z",
      "dataLen":570,
      "recordNumber":7,
      "tableID":"5AFA15123F991C43B6ACF8B46A914BD0",
```

```
      "tableName":"elem",
      "fields": {
            "1": 9,
            "2": "test value",
            "3": "2003-03-03T00:00:00.000Z",
            "4": "BlobPath: Table 1/Field 4/Data_9ACB28F1A2744FDFA5822B22F18B2E12.png",
            "8": "BlobID: 2"
      },
      "extraData": {
          "task_id": 1,
          "user_name": "Vanessa Smith",
          "user4d_id": 1,
          "host_name": "iMac-VSmith-0833",
          "task_name": "Application process",
          "client_version": -1610541776
      },
      "primaryKey": "9"
   }
]
```

Note : Si vous passez Attribut champ nom dans le paramètre *attribChamp*, l'objet "fields" contiendra :

```
...
      "fields": {
            "ID": 9,
            "Field_2": "test value",
            "Date_Field": "2003-03-03T00:00:00.000Z",
            "Field_4": "BlobPath: Table 1/Field 4/Data_9ACB28F1A2744FDFA5822B22F18B2E12.png",
            "Field_8": "BlobID: 2"
      },...
```

La liste effective des propriétés dépend du type d'opération (i.e.: création, suppression ou modification de l'enregistrement, créer Blob, etc.). Les principales propriétés sont les suivantes :

- *operationType* : Code interne de l'opération
- *operationName* : Type d'opération, par exemple "create record," "modify record"
- *operationNumber* : Numéro interne de l'opération dans le fichier d'historique
- *contextID* : ID du contexte d'exécution ; le contexte est détaillé dans la section *extraData*
- *timeStamp* : horodatage de l'opération dans le fichier d'historique
- *dataLen* : taille interne des données
- *recordNumber* : numéro interne d'enregistrement
- *tableID* : ID interne de la table
- *tableName* : nom de la table
- *fields* : objet contenant la liste des numéros de champs (ou des noms de champs) ainsi que leur valeur. Tous les champs de la table dont la valeur a été modifiée sont listés. Dans le cas de valeurs de type Blob ou image, différentes informations sont stockées en fonction de leur mode de stockage :
  - si le Blob ou l'image est stocké(e) dans le fichier de données, la propriété sera "BlobID:"+ un numéro de Blob interne, par exemple : "BlobID:1"
  - si le Blob ou l'image est stocké(e) à l'extérieur du fichier de données, la propriété sera "BlobPath:" + chemin du fichier, par exemple : "BlobPath: Table 1/Field 6/Data_EE12D091535F9748BCE62EDE972A4BA2.jpg"
- *extraData* : données du contexte de l'utilisateur, incluant son nom et son ID, le nom et l'ID de la tâche, le nom de la machine hôte ainsi que la version du client.
- *sequenceNumber* : numéro courant au sein d'une séquence d'incrémentation automatique.
- *primaryKey* : valeur de clé primaire.

Exemple

Vous voulez sauvegarder le fichier d'historique courant en JSON :

```
FICHIER HISTORIQUE VERS JSON("c:\\4Dv15\\ExportLogs")
```

Vous voulez sauvegarder un fichier d'historique spécifique en JSON avec les champs identifiés par nom :

```
FICHIER HISTORIQUE VERS JSON("c:\\4Dv15\\ExportLogs";0;"c:\\4Dv15\\Backup\\old_myDB.journal";Attribut champ nom)
```

Variables et ensembles système

La commande **FICHIER HISTORIQUE VERS JSON** modifie la valeur des variables système OK et Document : si le fichier JSON a été correctement sauvegardé, OK prend la valeur 1 et Document contient le chemin du fichier d'historique. Si vous avez passé "" dans le paramètre *logPath* et que l'utilisateur a annulé la boîte de dialogue de sélection de fichier, OK prend la valeur 0 et Document contient une chaîne vide. Si l'utilisateur a sélectionné un fichier invalide, OK prend la valeur 0 et Document contient le chemin du fichier invalide.

FIXER FICHIER HISTORIQUE

FIXER FICHIER HISTORIQUE (historique)

Paramètre	Type		Description
historique	Opérateur, Chaîne	⇨	Nom du fichier d'historique ou * pour refermer l'historique courant

Description

La commande **FIXER FICHIER HISTORIQUE** crée ou ferme le fichier d'historique de la base de données, suivant la valeur que vous passez dans *historique*.

Note : Appeler la commande **FIXER FICHIER HISTORIQUE** équivaut à sélectionner/désélectionner l'option **Utiliser le fichier d'historique...** dans la page **Sauvegarde**/**Configuration** des Préférences de l'application.

Passez dans *historique* le nom ou le chemin d'accès complet du fichier d'historique à créer. Si vous passez uniquement un nom, le fichier sera créé dans le dossier "Logs" de la base, à côte du fichier de structure de la base. Si vous passez une chaîne vide, **FIXER FICHIER HISTORIQUE** présente une boîte de dialogue standard d'enregistrement de fichier, permettant à l'utilisateur de choisir le nom et l'emplacement du fichier d'historique à créer. Si le fichier est correctement créé, la variable **OK** prend la valeur 1. Autrement, si l'utilisateur clique sur le bouton **Annuler** ou si le fichier d'historique ne peut pas être créé, **OK** prend la valeur 0.

Note : Le nouveau fichier d'historique n'est pas généré immédiatement après l'exécution de la commande, mais après la prochaine sauvegarde (le paramètre est conservé dans le fichier de données, il sera pris en compte même si la base est refermée entre-temps. Vous pouvez appeler la commande **SAUVEGARDER** pour provoquer la création du fichier d'historique.

Si vous passez * dans *historique*, **FIXER FICHIER HISTORIQUE** referme le fichier d'historique courant de la base. La variable **OK** prend la valeur 1 lorsque le fichier d'historique est refermé.

Si vous utilisez **FIXER FICHIER HISTORIQUE** pour créer un fichier d'historique avant qu'une sauvegarde n'ait été réalisée et si le fichier de données contient déjà des enregistrements, 4D génère l'erreur *-4447*, que vous pouvez intercepter avec une méthode installée par **APPELER SUR ERREUR**.

Variables et ensembles système

OK prend la valeur 1 si le fichier d'historique est correctement créé ou fermé.

Gestion des erreurs

L'erreur -4447 est générée si l'opération ne peut pas être réalisée car la base de données doit être auparavant sauvegardée.

INTEGRER FICHIER HISTORIQUE MIROIR

INTEGRER FICHIER HISTORIQUE MIROIR (cheminAccès ; numOpération {; mode {; objErreur}})

Paramètre	Type		Description
cheminAccès	Texte	⇒	Nom ou chemin d'accès du fichier d'historique à intégrer
numOpération	Variable réel	⇒	Numéro de la dernière opération intégrée ou -2 pour tout intégrer
		⇐	Nouveau numéro de la dernière opération intégrée
mode	Entier long	⇒	0=mode strict (mode par défaut), 1=mode réparation auto
objErreur	Variable objet	⇐	Opération(s) manquante(s)

Description

Note préliminaire : Cette commande ne fonctionne qu'avec 4D Server. Elle ne peut être exécutée que via la commande **Executer sur serveur** ou dans une procédure stockée.

La commande **INTEGRER FICHIER HISTORIQUE MIROIR** permet d'intégrer le fichier d'historique désigné par *cheminAccès* dans une base 4D Server, à la suite de l'opération *numOpération*. La commande accepte d'intégrer tout fichier d'historique dans la base, même s'il ne correspond pas au fichier de données. Cette commande est particulièrement destinée à une utilisation dans le contexte d'une base miroir.

Note : Depuis 4D v14, il est possible d'utiliser un fichier d'historique dans le contexte d'une base miroir : l'option "Utiliser fichier d'historique" peut désormais être cochée dans les propriétés d'une base 4D Server utilisée comme miroir logique, permettant la mise en place de serveurs miroirs en série (cf. section **Mise en place d'un miroir logique** dans le manuel de 4D Server).

A la différence de la commande **_o_INTEGRER FICHIER HISTORIQUE**, la commande **INTEGRER FICHIER HISTORIQUE MIROIR** ne substitue pas le fichier d'historique intégré à l'historique courant à l'issue de son exécution : le fichier d'historique de la base continue d'être utilisé. Par conséquent, les modifications effectuées lors de l'intégration sont enregistrées dans le fichier d'historique courant.

Passez dans *cheminAccès* un chemin d'accès absolu ou relatif au dossier de la base. Si vous passez une chaîne vide dans ce paramètre, une boîte de dialogue standard d'ouverture de fichier s'affiche, permettant de désigner le fichier à intégrer. Si la boîte de dialogue est annulée, aucun fichier n'est intégré et la variable système *OK* prend la valeur 0.

Passez dans la variable *numOpération* le numéro de la dernière opération intégrée, afin que l'intégration débute à l'opération suivante. A l'issue de l'intégration, la valeur de la variable *numOpération* est mise à jour avec le numéro de la dernière opération intégrée. Vous devez alors stocker cette variable puis la réutiliser directement comme paramètre *numOpération* lors de l'opération d'intégration suivante. Ce principe vous permet d'enchaîner les intégrations d'historiques à l'aide de **INTEGRER FICHIER HISTORIQUE MIROIR**. Passez la valeur -2 dans la variable si vous souhaitez intégrer la totalité du fichier.

Note de compatibilité : *Dans les versions de 4D antérieures à la v15 R4, le paramètre numOpération était optionnel. Désormais, si le paramètre numOpération est omis, une erreur est générée. Pour rétablir le fonctionnement initial de votre ancien code, passez simplement une variable avec la valeur -2 dans le paramètre numOpération.*

Le paramètre *mode* vous permet de spécifier le mode d'intégration que vous souhaitez utiliser. Vous pouvez passer une des constantes suivantes, placées dans le thème "**Sauvegarde et restitution**" :

Constante	Type	Valeur	Comment
Mode réparation auto	Entier long	1	Utiliser le mode flexible avec réparation automatique et remplir le paramètre *objErreur* (si passé)
Mode strict	Entier long	0	Utiliser le mode d'intégration avec contrôle strict des opérations (option par défaut). Recommandé dans la plupart des cas.

- Mode strict : Dans ce mode, dès qu'une erreur se produit au cours de l'intégration, la procédure est stoppée et vous devez utiliser le CSM pour tracer l'erreur. Ce mode sécurisé est utilisé par défaut et est recommandé dans la plupart des cas.
- Mode réparation auto : Dans ce mode, lorsqu'une erreur non critique est détectée, elle est "réparée" et l'intégration se poursuit. Si vous avez passé le paramètre *objErreur*, chaque erreur est enregistrée et pourra être analysée par la suite.
  Les cas d'erreurs non critiques sont les suivants :
  - Le fichier d'historique demande à ajouter un enregistrement mais l'enregistrement existe déjà dans les données.
    Action de réparation : 4D met à jour l'enregistrement.
  - Le fichier d'historique demande à mettre à jour un enregistrement mais l'enregistrement n'existe pas.
    Action de réparation : 4D ajoute l'enregistrement.
  - Le fichier d'historique demande à supprimer un enregistrement mais l'enregistrement n'existe pas.
    Action de réparation : 4D ne fait rien.

Note : En mode strict (mode par défaut), l'intégration stoppe à la première erreur rencontrée. Si vous souhaitez poursuivre l'intégration dans ce cas, il sera nécessaire d'utiliser le CSM.

Lorsqu'une anomalie se produit en mode réparation auto, l'enregistrement concerné est automatiquement "réparé" et l'opération associée est enregistrée dans le paramètre *objErreur*. Une fois l'intégration terminée, le paramètre *objErreur* contient la liste de tous les enregistrements réparés. Il se compose d'un unique tableau d'objets nommé "operations", structuré de la manière suivante :

```
{"operations":
    [
        {
            "operationType":24,
            "operationName":"Create record",
            "operationNumber":2,
            "contextID":48,
            "timeStamp":"2015-07-10T07:53:02.413Z",
            "dataLen":24,
            "recordNumber":0,
            "tableID":"F4CXXXXX",
            "tableName":"Customers",
            "fields": {
```

```
                "1": 9,
                "2": "test value",
                "3": "2003-03-03T00:00:00.000Z",
                "4": "BlobPath: Table 1/Field 4/Data_9ACB28F1A2744FDFA5822B22F18B2E12.png",
                "8": "BlobID: 2"
            }
        },
        {...}
    ]
```

Attention : Le mode réparation auto doit être activé dans des cas spécifiques car il contourne les sécurités intégrées de 4D chargées de contrôler l'intégrité des données. Il peut être utilisé, par exemple, lorsqu'un fichier d'historique intermédiaire a été perdu ou corrompu et que vous souhaitez récupérer autant d'opérations que possible. Dans tous les cas, vous devez être particulièrement vigilant en ce qui concerne l'intégrité des données lorsque ce mode est activé.

La liste effective des propriétés présentes dans l'objet "operations" dépend du type d'opération (i.e.: création, suppression ou modification de l'enregistrement, créer Blob, etc.). Les principales propriétés sont les suivantes :

- *operationType* : Code interne de l'opération
- *operationName* : Type d'opération, par exemple "create record," "modify record"
- *operationNumber* : Numéro interne de l'opération dans le fichier d'historique
- *contextID* : ID du contexte d'exécution ; le contexte est détaillé dans la section *extraData*
- *timeStamp* : horodatage de l'opération dans le fichier d'historique
- *dataLen* : taille interne des données
- *recordNumber* : numéro interne d'enregistrement
- *tableID* : ID interne de la table
- *tableName* : nom de la table
- *fields* : objet contenant la liste des numéros de champ ainsi que leur valeur. Tous les champs de la table sont listés.
  Dans le cas de valeurs de type Blob ou image, différentes informations sont stockées en fonction de leur mode de stockage :
  - si le Blob ou l'image est stocké(e) dans le fichier de données, la propriété sera "BlobID:"+ un numéro de Blob interne, par exemple : "BlobID:1"
  - si le Blob ou l'image est stocké(e) à l'extérieur du fichier de données, la propriété sera "BlobPath:" + chemin du fichier, par exemple : "BlobPath: Table 1/Field 6/Data_EE12D091535F9748BCE62EDE972A4BA2.jpg"
- *extraData* : données du contexte de l'utilisateur, incluant son nom et son ID, le nom et l'ID de la tâche, le nom de la machine hôte ainsi que la version du client.
- *sequenceNumber* : numéro courant au sein d'une séquence d'incrémentation automatique.
- *primaryKey* : valeur de clé primaire.

Exemple

Vous voulez intégrer un fichier d'historique sur le serveur miroir en mode réparation auto :

```
 //à exécuter sur le serveur
C_OBJET($err)
C_ENTIER LONG($num) //-2 pour tout intégrer
INTEGRER FICHIER HISTORIQUE MIROIR("c:\\mirror\\logNew.journal";$num;Mode réparation auto;$err)
```

Variables et ensembles système

Si l'intégration s'effectue correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

LIRE INFORMATION RESTITUTION

LIRE INFORMATION RESTITUTION (sélecteur ; info1 ; info2)

Paramètre	Type		Description
sélecteur	Entier long	⇨	Type d'information à récupérer
info1	Entier long, Date	⇦	Valeur 1 du sélecteur
info2	Chaîne, Heure	⇦	Valeur 2 du sélecteur

Description

La commande **LIRE INFORMATION RESTITUTION** permet de récupérer des informations relatives à la dernière restitution automatique de la base.

Passez dans le paramètre *sélecteur* le type d'information à récupérer. Vous pouvez utiliser une des constantes suivantes, placées dans le thème "**Sauvegarde et restitution**" :

Constante	Type	Valeur
Date dernière restitution	Entier long	0
Statut dernière restitution	Entier long	2

Le type et le contenu des paramètres *info1* et *info2* dépendent de la valeur de *sélecteur*.

- Si *sélecteur* = 0 (Date dernière restitution), *info1* retourne la date et *info2* l'heure de la dernière restitution automatique de la base.
- Si *sélecteur* = 2 (Statut dernière restitution), *info1* retourne le numéro et *info2* le texte du statut de la dernière restitution automatique de la base.

Note : Cette commande ne tient pas compte des restitutions manuelles de la base.

LIRE INFORMATION SAUVEGARDE

LIRE INFORMATION SAUVEGARDE (sélecteur ; info1 ; info2)

Paramètre	Type		Description
sélecteur	Entier long	⇨	Type d'information à récupérer
info1	Entier long, Date	⇦	Valeur 1 du sélecteur
info2	Heure, Chaîne	⇦	Valeur 2 du sélecteur

Description

La commande **LIRE INFORMATION SAUVEGARDE** permet de récupérer des informations relatives à la dernière sauvegarde effectuée sur les données de la base.

Passez dans le paramètre *sélecteur* le type d'information à récupérer. Vous pouvez utiliser une des constantes suivantes, placées dans le thème "**Sauvegarde et restitution**" :

Constante	Type	Valeur
Date dernière sauvegarde	Entier long	0
Date prochaine sauvegarde	Entier long	4
Statut dernière sauvegarde	Entier long	2

Le type et le contenu des paramètres *info1* et *info2* dépendent de la valeur de *sélecteur*.

- Si *sélecteur* = 0 (Date dernière sauvegarde), *info1* retourne la date et *info2* l'heure de la dernière sauvegarde.
- Si *sélecteur* = 2 (Statut dernière sauvegarde), *info1* retourne le numéro et *info2* le texte du statut de la dernière sauvegarde.
- Si *sélecteur* = 4 (Date prochaine sauvegarde), *info1* retourne la date et *info2* l'heure de la prochaine sauvegarde prévue.

Nouveau fichier historique

Nouveau fichier historique -> Résultat

Paramètre	Type		Description
Résultat	Texte		Chemin d'accès complet du fichier d'historique refermé

Description

Note préliminaire : Cette commande ne fonctionne qu'avec 4D Server. Elle ne peut être exécutée que via la commande **Executer sur serveur** ou dans une procédure stockée.

La commande **Nouveau fichier historique** referme le fichier d'historique courant, le renomme et en crée un nouveau avec le même nom et au même emplacement que le précédent. Cette commande est destinée à la mise en place d'un système de sauvegarde par miroir logique (cf. section "**Mise en place d'un miroir logique**" dans le Manuel de référence de 4D Server).

La commande retourne le nom complet (chemin d'accès+nom) du fichier d'historique refermé (appelé "segment"). Ce fichier est stocké au même emplacement que le fichier d'historique courant (spécifié dans la page Configuration des préférences de l'application, thème Sauvegarde). La commande n'effectue aucun traitement (compression, segmentation) sur le fichier sauvegardé. Aucune boîte de dialogue n'apparaît.

Le fichier est renommé avec les numéros de sauvegarde courants de la base et du fichier d'historique, sur le modèle suivant : *NomBase[NumSvgde-NumSvgdeHisto].journal*. Par exemple :

- si la base MaBase.4DD a été sauvegardée 4 fois, le dernier fichier de sauvegarde se nomme *MaBase[0004].4BK*. Le nom du premier "segment" de fichier d'historique sera donc *MaBase[0004-0001].journal*.
- si la base MaBase.4DD a été sauvegardée 3 fois et que le fichier d'historique a été sauvegardé 5 fois depuis, le nom de la 6e sauvegarde du fichier d'historique sera *MaBase[0003-0006].journal*.

Gestion des erreurs

En cas d'anomalie, la commande génère une erreur que vous pouvez intercepter à l'aide de la commande **APPELER SUR ERREUR**.

RESTITUER

RESTITUER {(cheminArchive {; cheminDossierDest})}

Paramètre	Type		Description
cheminArchive	Texte	⇒	Chemin d'accès de l'archive à restituer
cheminDossierDest	Texte	⇒	Chemin d'accès du dossier de destination

Description

La commande **RESTITUER** permet restituer le ou les fichier(s) inclus dans une archive 4D. Cette commande est utile dans le cadre d'interfaces personnalisées pour la gestion des sauvegardes.

Si vous ne passez pas le paramètre *cheminArchive*, la commande affiche une boîte de dialogue d'ouverture permettant à l'utilisateur de sélectionner l'archive à restituer.

Le paramètre *cheminArchive* vous permet d'indiquer le chemin d'accès du fichier d'archive à restituer. Ce chemin doit être exprimé avec la syntaxe système. Vous pouvez passer un chemin d'accès absolu ou relatif au fichier de structure de la base.
Dans ce cas (si le paramètre *cheminDossierDest* est omis), la boîte de dialogue de restitution standard apparaît avec l'archive pré-sélectionnée, permettant à l'utilisateur de désigner le dossier de destination. A l'issue de la procédure, une boîte de dialogue d'alerte apparaît et le dossier contenant les éléments restitués est affiché.

Vous pouvez également passer le paramètre *cheminDossierDest* avec le chemin d'accès du dossier de destination des éléments restitués. Ce chemin doit être exprimé avec la syntaxe système. Vous pouvez passer un chemin d'accès absolu ou relatif au fichier de structure de la base. Si vous passez ce paramètre, une boîte de dialogue de restitution préconfigurée apparaît, permettant uniquement à l'utilisateur de lancer ou d'annuler la restitution. A l'issue de la procédure, la fenêtre est simplement refermée sans affichage d'information supplémentaire.

La commande **RESTITUER** modifie la valeur des variables *OK* et *Document* : si la restitution s'est déroulée correctement, *OK* prend la valeur 1 et *Document* contient le chemin du dossier de restitution. Si l'utilisateur a annulé la boîte de dialogue de restitution, interrompu la restitution ou si une erreur s'est produite, *OK* prend la valeur 0 et *Document* contient une chaîne vide. Vous pouvez intercepter l'erreur à l'aide d'une méthode installée via la commande **APPELER SUR ERREUR**.

Note : Dans le cadre d'une application 4D compilée et fusionnée avec 4D Volume Desktop, la commande **RESTITUER** provoque l'affichage d'une boîte de dialogue système standard d'ouverture de fichiers, proposant par défaut les fichiers d'extension "4BK".

SAUVEGARDER

SAUVEGARDER
Ne requiert pas de paramètre

Description

La commande **SAUVEGARDER** déclenche la sauvegarde de la base de données avec les paramétrages de sauvegarde courants. Aucune boîte de dialogue de confirmation n'est affichée. Une fenêtre de progression apparaît à l'écran.

Les paramétrages de sauvegarde sont définis dans les Propriétés de la base ("Préférences" dans 4D v11 SQL). Ils sont également stockés dans le fichier Backup.XML situé dans le sous-dossier Preferences/Backup de la base.

La commande **SAUVEGARDER** appelle la **Méthode base Sur démarrage sauvegarde** au début de son exécution et la **Méthode base Sur arrêt sauvegarde** à la fin de son exécution.
Attention, du fait de ce mécanisme, la commande ne doit PAS être appelée depuis l'une de ces méthodes base.

4D Server : Lorsqu'elle est appelée depuis un poste client, la commande **SAUVEGARDER** est considérée comme une procédure stockée, elle est toujours exécutée sur le serveur.

Variables et ensembles système

Si la sauvegarde se déroule correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Gestion des erreurs

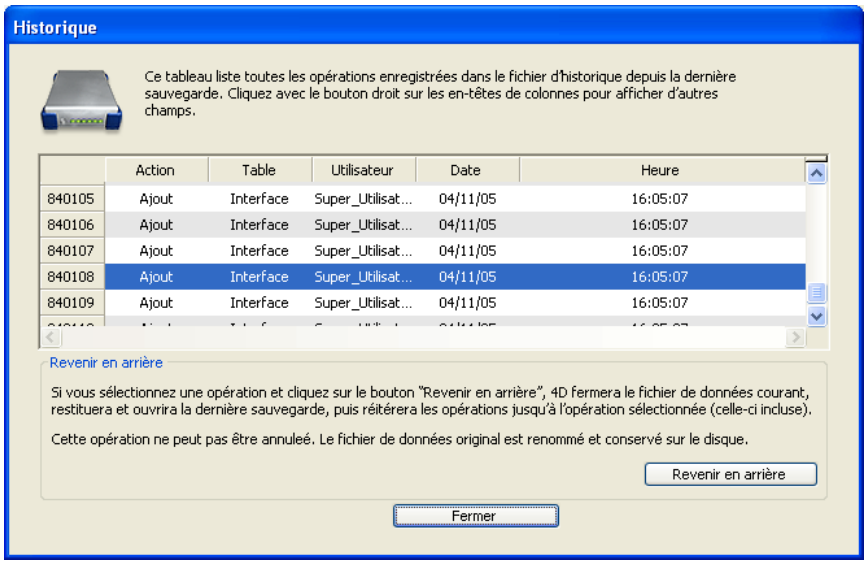
En cas d'incident au cours de la sauvegarde, les informations relatives à l'incident sont écrites dans le journal des sauvegardes et l'erreur de plus haut niveau est transmise uniquement à la **Méthode base Sur arrêt sauvegarde**. Il est donc particulièrement important d'utiliser cette méthode base afin de pouvoir gérer par programmation les erreurs liées à la sauvegarde.

VERIFIER FICHIER HISTORIQUE

VERIFIER FICHIER HISTORIQUE
Ne requiert pas de paramètre

Description

La commande **VERIFIER FICHIER HISTORIQUE** affiche la boîte de dialogue de visualisation du fichier d'historique courant de la base (également accessible via la fenêtre du Centre de sécurité et de maintenance) :



Cette boîte de dialogue comporte le bouton **Revenir en arrière**, permettant d'annuler des opérations effectuées sur les données de la base. Pour plus d'informations sur cette boîte de dialogue, reportez-vous au manuel Mode Développement de 4D.

Note : La fonction de retour en arrière étant relativement puissante, il est conseillé de restreindre l'accès à la commande **VERIFIER FICHIER HISTORIQUE** aux administrateurs de la base.

Cette commande est utilisable dans le contexte d'une application monoposte uniquement. Elle permet notamment d'accéder à la fonction de retour en arrière depuis les applications 4D Volume Desktop (applications sans mode Développement). Si elle est appelée dans une application client/serveur, elle ne fait rien et l'erreur 1421 est retournée.

Gestion des erreurs

- Si cette commande est exécutée dans une base de données fonctionnant sans fichier d'historique, elle ne fait rien et l'erreur 1403 est retournée.
- Si cette commande est exécutée sur une base client/serveur, elle ne fait rien et l'erreur 1421 est retournée.
  Vous pouvez intercepter ces erreurs à l'aide d'une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

_o_INTEGRER FICHIER HISTORIQUE

_o_INTEGRER FICHIER HISTORIQUE (cheminAccès)

Paramètre	Type		Description
cheminAccès	Texte	⇨	Nom ou chemin d'accès du fichier d'historique à intégrer

Note de compatibilité

La commande **_o_INTEGRER FICHIER HISTORIQUE** est déclarée obsolète dans 4D à compter de la version 16 et est conservée pour des raisons de compatibilité uniquement. La fonctionnalité de sauvegarde par miroir logique de 4D s'appuie désormais uniquement sur la commande **INTEGRER FICHIER HISTORIQUE MIROIR**, qui a été optimisée et procure davantage de flexibilité.

Sélections

- ALLER A DERNIER ENREGISTREMENT
- ALLER DANS SELECTION
- APPLIQUER A SELECTION
- Avant selection
- CREER SELECTION SUR TABLEAU
- DEBUT SELECTION
- ENREGISTREMENT PRECEDENT
- ENREGISTREMENT SELECTION
- ENREGISTREMENT SUIVANT
- Enregistrements trouves
- Fin de selection
- LIRE ENREGISTREMENTS MARQUES
- MARQUER ENREGISTREMENTS
- MOBILE Renvoyer sélection
- MODIFIER SELECTION
- Numero dans selection
- Numero de ligne affichee
- REDUIRE SELECTION
- SCAN INDEX
- SUPPRIMER SELECTION
- TOUT SELECTIONNER
- VIDER TABLE
- VISUALISER SELECTION

ALLER A DERNIER ENREGISTREMENT

ALLER A DERNIER ENREGISTREMENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle vous voulez aller au dernier enregistrement ou Table par défaut si ce paramètre est omis

Description

ALLER A DERNIER ENREGISTREMENT désigne le dernier enregistrement de la sélection de *laTable* comme enregistrement courant et le charge en mémoire. Si la sélection est vide ou si l'enregistrement courant est déjà le dernier de la sélection, **ALLER A DERNIER ENREGISTREMENT** ne fait rien.

Exemple

L'exemple suivant désigne le dernier enregistrement de la table [Contacts] comme enregistrement courant :

```
ALLER A DERNIER ENREGISTREMENT([Contacts])
```

ALLER DANS SELECTION

ALLER DANS SELECTION ({laTable ;} enregistrement)

Paramètre	Type		Description
laTable	Table	⇒	Table dans laquelle aller à l'enregistrement spécifié ou Table par défaut si ce paramètre est omis
enregistrement	Entier long	⇒	Position de l'enregistrement dans la sélection

Description

La commande **ALLER DANS SELECTION** fait de l'enregistrement spécifié parmi la sélection courante de *laTable* l'enregistrement courant. La sélection courante n'est pas modifiée. Le paramètre *enregistrement* n'est pas équivalent au numéro retourné par **Numero enregistrement**. Ce paramètre représente la position de l'enregistrement au sein de la sélection courante. Cette position dépend de la manière dont la sélection a été créée et si elle a été triée.
Pour plus de précisions sur les numéros d'enregistrements, reportez-vous à la section **A propos des numéros d'enregistrements**.

ALLER DANS SELECTION ne fait rien si :

- il n'y a aucun enregistrement dans la sélection courante
- *enregistrement* n'appartient pas à la sélection courante,
- *enregistrement* est déjà l'enregistrement courant.

Si vous passez 0 dans *enregistrement*, il n'y a plus d'enregistrement courant dans *laTable*. Ce mécanisme permet de n'avoir plus aucun enregistrement sélectionné dans une liste, notamment dans les sous-formulaires inclus, lorsque le mode de sélection est "unique".

Exemple

L'exemple suivant charge les valeurs du champ [Personnes]Nom dans le tableau *taNoms*. Un tableau d'entiers longs, *numEnr*, est rempli avec des numéros qui représenteront ceux des enregistrements sélectionnés. Les deux tableaux sont alors triés :

```
  ` Créer ici la sélection de la table [Personnes]
  `...
  ` Récupérer les noms
SELECTION VERS TABLEAU([Personnes]Nom;taNoms)
  ` Créer un tableau pour les numéros d'enregistrements sélectionnés
$vELNbEnrgs:=Taille tableau(taNoms)
TABLEAU ENTIER LONG(numEnr;$vELNbEnrgs)
Boucle($Enrg;1;$vELNbEnrgs)  ` Remplir le tableau avec ces numéros
   numEnr{$Enrg}:=$Enrg
Fin de boucle
  ` Trier les deux tableaux par ordre alphabétique
TRIER TABLEAU(taNoms;numEnr;>)
```

Si le tableau *taNoms* est affiché dans une zone de défilement, l'utilisateur peut cliquer sur l'un des éléments. Comme les deux tableaux ont été triés de manière synchronisée, tout élément de *numEnr* fournit le numéro de l'enregistrement sélectionné pour lequel le nom a été stocké dans l'élément de *taNoms* correspondant.

La méthode objet de la zone de défilement *taNoms* suivante sélectionne le bon enregistrement dans la sélection de [Personnes] en fonction de ce que l'utilisateur a choisi dans la zone de défilement.

```
Au cas ou
   :(Evenement formulaire=Sur clic)
      Si(taNoms#0)
         ALLER DANS SELECTION(numEnr{taNoms})
      Fin de si
Fin de cas
```

APPLIQUER A SELECTION

APPLIQUER A SELECTION (laTable ; formule)

Paramètre	Type		Description
laTable	Table	➡	Table dans laquelle appliquer la formule
formule	Instruction	➡	Ligne de code ou méthode

Description

La commande **APPLIQUER A SELECTION** applique *formule* à chaque enregistrement de la sélection courante de *laTable*. La *formule* peut être une ligne d'instructions ou une méthode (dans ce cas, le nom de la méthode doit être saisi sans ""). Si *formule* entraîne la modification d'un enregistrement de *laTable*, l'enregistrement modifié est sauvegardé. Si *formule* ne modifie pas d'enregistrement, aucune sauvegarde n'est réalisée. Si la sélection courante est vide, **APPLIQUER A SELECTION** ne fait rien. La *formule* peut faire appel à un champ d'une table liée si le lien est automatique.

La commande **APPLIQUER A SELECTION** peut être utilisée pour récupérer et traiter des informations sur une sélection d'enregistrements (par exemple, calcul d'un total), ou pour modifier une sélection (par exemple, mettre en majuscule la première lettre d'un champ). Si cette commande est utilisée à l'intérieur d'une transaction, toutes les opérations réalisées pourront être annulées si la transaction n'est pas validée.

4D Server : Le serveur n'exécute aucune des commandes passées dans *formule*. Chaque enregistrement de la sélection est renvoyé sur le poste client pour traitement et modification.

Un thermomètre de progression s'affiche pendant l'exécution d'un **APPLIQUER A SELECTION**. Un appel préalable à la commande **SUPPRIMER MESSAGES** permet de supprimer ce thermomètre. Lorsque le thermomètre de progression est affiché, l'utilisateur peut annuler l'opération.

Exemple 1

L'exemple suivant met en majuscule tous les noms de la table :

```
APPLIQUER A SELECTION([Emp];[Emp]Nom:=Majusc([Emp]Nom))
```

Exemple 2

Lorsque **APPLIQUER A SELECTION** rencontre un enregistrement verrouillé et le modifie, celui-ci n'est pas sauvegardé. Tous les enregistrements verrouillés rencontrés par la commande sont placés dans un ensemble système appelé *LockedSet*. Après l'exécution d'un **APPLIQUER A SELECTION**, il est recommandé de tester l'ensemble *LockedSet* pour vérifier la présence d'enregistrements verrouillés. La boucle suivante s'exécute jusqu'à ce que tous les enregistrements aient été modifiés :

```
Repeter ` Pour chaque enregistrement verrouillé
   APPLIQUER A SELECTION([Emp];[Emp]Nom:=Majusc([Emp]Nom))
   UTILISER ENSEMBLE("LockedSet") ` Sélection des enregistrements verrouillés uniquement
 ` Jusqu'à ce qu'il n'y ait plus d'enregistrement verrouillé
Jusque(Enregistrements dans ensemble("LockedSet")=0)
```

Exemple 3

Cet exemple utilise une méthode :

```
TOUT SELECTIONNER([Emp])
APPLIQUER A SELECTION([Emp];Capitales)
```

Variables et ensembles système

Si l'utilisateur clique sur le bouton Stop dans le thermomètre de progression, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.

Avant selection

Avant selection {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table pour laquelle vous testez si le pointeur se trouve avant la sélection
Résultat	Booléen	⇐	Avant sélection (Vrai) sinon (Faux)

Description

La fonction **Avant selection** retourne **Vrai** lorsque le pointeur d'enregistrement courant se trouve avant le premier enregistrement de la sélection courante de *laTable*. **Avant selection** est généralement utilisée pour vérifier si la commande **ENREGISTREMENT PRECEDENT** a déplacé le pointeur d'enregistrement courant avant le premier enregistrement. Si la sélection courante est vide, **Avant selection** retourne **Vrai**.

Pour replacer le pointeur d'enregistrement courant dans la sélection courante, utilisez les commandes **DEBUT SELECTION**, **ALLER A DERNIER ENREGISTREMENT** ou **ALLER DANS SELECTION**. **ENREGISTREMENT SUIVANT** ne replace pas le pointeur d'enregistrement courant dans la sélection courante.

Avant selection retourne **Vrai** dans l'en-tête lorsqu'un état est en cours d'impression à l'aide de la commande **IMPRIMER SELECTION** ou à partir de la commande de menu Imprimer. Vous pouvez utiliser le code suivant pour tester le premier en-tête et imprimer un en-tête spécial pour la première page :

```
  ` Méthode d'un formulaire sortie utilisé pour un état
$vpFormTable:=Table du formulaire courant
Au cas ou
  ` ...
   :(Evenement formulaire=Sur entête)
  ` La zone en-tête va être imprimée
      Au cas ou
         :(Avant selection($vpFormTable->))
  ` Le code pour la première rupture d'en-tête doit être placé ici
  ` ...
      Fin de cas
Fin de cas
```

Exemple

La méthode formulaire suivante est utilisée pendant l'impression d'un état. Elle définit une variable *vTitre* à imprimer dans la zone d'en-tête sur la première page :

```
  ` Méthode formulaire [Finances];"Tableau"
Au cas ou
  ` ...
   :(Evenement formulaire=Sur entête)
  ` La zone en-tête va être imprimée
      Au cas ou
         :(Avant selection([Finances]))
            vTitre:="Etat des finances pour 1997" ` Définir le titre pour la première page
         Sinon
            vTitre:="" ` Effacer le titre pour les autres pages
      Fin de cas
Fin de cas
```

CREER SELECTION SUR TABLEAU

CREER SELECTION SUR TABLEAU (laTable ; tabEnrg {; nom})

Paramètre	Type		Description
laTable	Table	⇒	Table de la sélection
tabEnrg	Entier long, Tableau booléen	⇒	Tableau de n° d'enregistrements, ou Tableau de booléens (Vrai = l'enregistrement est dans la sélection, Faux = il n'est pas dans la sélection)
nom	Chaîne	⇒	Nom de la sélection temporaire à créer, ou Appliquer la commande à la sélection courante si ce paramètre est omis ou vide

Description

La commande **CREER SELECTION SUR TABLEAU** construit la sélection temporaire *nom* à partir :

- soit du tableau de numéros d'enregistrements absolus *tabEnrg* de *laTable*,
- soit du tableau de booléens *tabEnrg* ; dans ce cas, les valeurs du tableau indiquent l'appartenance (**Vrai**) ou non (**Faux**) de chaque enregistrement de *laTable* à la sélection *nom*.

Si vous ne passez pas le paramètre *nom* ou si vous passez une chaîne vide, la commande s'appliquera à la sélection courante de *laTable*, qui sera donc mise à jour.

Lorsque vous utilisez la commande avec un tableau d'entiers longs, tous les numéros du tableau représentent la liste des numéros d'enregistrements qui feront partie de la sélection *nom*. Si un numéro est invalide (enregistrement non créé), l'erreur -10503 est générée.

Note : Attention, vous devez veiller à ce que le tableau ne contienne pas d'éléments ayant la même valeur, sinon la sélection résultante sera incorrecte.

Lorsque vous utilisez la commande avec un tableau de booléens, le Nième élément du tableau représente l'intégration (Vrai) ou non (Faux) de l'enregistrement numéro N dans la sélection *nom*. En principe, le nombre d'éléments du tableau doit être égal au nombre d'enregistrements de *laTable*. Si le tableau est plus petit que le nombre d'enregistrements, seuls les enregistrements définis par le tableau pourront faire partie de la sélection.

Note : Avec un tableau de booléens, la commande utilise les éléments du numéro 0 au numéro N-1.

Gestion des erreurs

Si un numéro d'enregistrement est invalide (enregistrement non créé), l'erreur -10503 est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

DEBUT SELECTION

DEBUT SELECTION {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle charger le premier enregistrement de la sélection courantes ou Table par défaut si ce paramètre est omis

Description

DEBUT SELECTION charge en mémoire le premier enregistrement de la sélection courante de *laTable* et en fait l'enregistrement courant. Toutes les commandes de recherche, de sélection et de tri font également du premier enregistement l'enregistrement courant. Si la sélection courante est vide ou si l'enregistrement courant est déjà le premier enregistrement de la sélection, **DEBUT SELECTION** ne fait rien.

Cette commande est principalement utilisée après un appel à **UTILISER ENSEMBLE**, pour débuter une boucle dans la sélection d'enregistrements à partir du premier enregistrement. Cependant, il est tout à fait envisageable de l'appeler depuis une sous-routine lorsque vous souhaitez vous assurer que l'enregistrement est bien le premier.

Exemple

L'exemple suivant charge le premier enregistrement de la table [Clients] :

```
DEBUT SELECTION([Clients])
```

ENREGISTREMENT PRECEDENT

ENREGISTREMENT PRECEDENT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table dans laquelle se placer sur l'enregistrement précédent de la sélection courante ou Table par défaut si ce paramètre est omis

Description

ENREGISTREMENT PRECEDENT place le pointeur d'enregistrement courant sur l'enregistrement précédent dans la sélection courante de *laTable* pour le process courant. Si la sélection courante est vide, ou si **Avant selection** ou **Fin de selection** renvoie **Vrai**, **ENREGISTREMENT PRECEDENT** ne fait rien.

Si **ENREGISTREMENT PRECEDENT** place le pointeur d'enregistrement courant avant la sélection courante, **Fin de selection** retourne **Vrai**, et il n'y a plus d'enregistrement courant. Dans ce cas, utilisez les commandes **DEBUT SELECTION**, **ALLER A DERNIER ENREGISTREMENT** ou **ALLER DANS SELECTION** pour replacer le pointeur d'enregistrement courant dans la sélection courante.

ENREGISTREMENT SELECTION

ENREGISTREMENT SELECTION {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle réduire la sélection à un enregistrement

Description

La commande **ENREGISTREMENT SELECTION** réduit la sélection courante de *laTable* à l'enregistrement courant. S'il n'y a pas d'enregistrement courant ou si l'enregistrement courant n'est pas chargé en mémoire (cas particulier), **ENREGISTREMENT SELECTION** ne fait rien.

Note

A l'origine, cette commande était utile pour "replacer" dans la sélection courante un enregistrement qui avait été empilé puis dépilé de la pile d'enregistrements pendant que la sélection de la table était modifiée. Cependant, puisque la commande **FIXER DESTINATION RECHERCHE** vous permet d'effectuer une recherche sans changer la sélection ni l'enregistrement courants de la table, vous n'avez plus besoin d'empiler et de dépiler un enregistrement courant pour effectuer une recherche sur sa table. Par conséquent, **ENREGISTREMENT SELECTION** est moins utile, à moins que vous ne souhaitiez expressément, pour une autre raison, réduire la sélection d'une table à l'enregistrement courant.

ENREGISTREMENT SUIVANT

ENREGISTREMENT SUIVANT {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table dans laquelle se placer sur l'enregistrement suivant ou Table par défaut si ce paramètre est omis

Description

La commande **ENREGISTREMENT SUIVANT** place le pointeur d'enregistrement courant sur l'enregistrement suivant dans la sélection courante de *laTable* pour le process courant. Si la sélection courante est vide, ou si **Avant selection** ou **Fin de selection** retourne **Vrai**, **ENREGISTREMENT SUIVANT** ne fait rien.

Si **ENREGISTREMENT SUIVANT** place le pointeur d'enregistrement courant après la fin de la sélection courante, **Fin de selection** retourne **Vrai**, et il n'y a alors plus d'enregistrement courant. Lorsque **Fin de selection** retourne **Vrai**, utilisez les commandes **DEBUT SELECTION**, **ALLER A DERNIER ENREGISTREMENT** ou **ALLER DANS SELECTION** pour replacer le pointeur d'enregistrement courant dans la sélection courante.

Exemple

Reportez-vous à l'exemple de la commande **AFFICHER ENREGISTREMENT**.

Enregistrements trouves

Enregistrements trouves {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table dont vous souhaitez connaître le nombre d'enregistrements de la sélection courante ou Table par défaut si ce paramètre est omis
Résultat	Entier long	⇐	Nombre d'enregistrements dans la sélection courante de table

Description

Enregistrements trouves retourne le nombre d'enregistrements constituant la sélection courante de *laTable* (par opposition, **Enregistrements dans table** retourne le nombre total d'enregistrements d'une table).

Exemple

L'exemple suivant propose une technique de boucle couramment utilisée pour se déplacer parmi les enregistrements de la sélection courante. La même opération peut être réalisée à l'aide de la commande **APPLIQUER A SELECTION** :

```
DEBUT SELECTION([Personnes]) ` Départ sur le premier enregistrement de la sélection
Boucle($VElEnreg;1;Enregistrements trouves([Personnes])) ` Boucle une fois par enregistrement
   Faire quelque chose ` Réaliser une opération avec l'enregistrement
   ENREGISTREMENT SUIVANT([Personnes]) ` Passage à l'enregistrement suivant
Fin de boucle
```

Fin de selection

Fin de selection {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table pour laquelle tester si le pointeur d'enregistrement courant est au-delà du dernier enregistrement de la sélection courante ou Table par défaut si ce paramètre est omis
Résultat	Booléen	⇐	Oui (Vrai), Non (Faux)

Description

La fonction **Fin de selection** retourne **Vrai** lorsque le pointeur de l'enregistrement courant se trouve après le dernier enregistrement de la sélection courante de *laTable*. **Fin de selection** est généralement utilisée pour tester si l'appel à la commande **ENREGISTREMENT SUIVANT** place ou non le pointeur d'enregistrement courant derrière le dernier enregistrement de la sélection. Si la sélection courante est vide, **Fin de selection** retourne **Vrai**.

Pour replacer le pointeur d'enregistrement courant dans la sélection, utilisez les commandes **DEBUT SELECTION**, **ALLER A DERNIER ENREGISTREMENT** ou **ALLER DANS SELECTION**. **ENREGISTREMENT PRECEDENT** ne replace pas le pointeur dans la sélection.

Fin de selection retourne également **Vrai** lors de l'impression du dernier pied de page d'un état, déclenchée par la commande **IMPRIMER SELECTION** ou le menu **Imprimer**. Vous pouvez utiliser l'instruction suivante pour intercepter le dernier pied de page et insérer une mention particulière :

```
  ` Méthode d'un formulaire sortie utilisé pour imprimer un état
$vpFormTable:=Table du formulaire courant
Au cas ou
  ` ...
  :(Evenement formulaire=Sur impression pied de page)
  ` Un pied
     Si(Fin de selection($vpFormTable->))
  ` Le code pour le dernier pied de page doit être placé ici
     Sinon
  ` Le code pour le pied de page doit être placé ici
     Fin de si
Fin de cas
```

Exemple

La méthode formulaire de l'exemple suivant est utilisée lors de l'impression d'un état. Elle crée la variable **VPied**, à imprimer dans le pied de page de la dernière page :

```
  ` Méthode formulaire [Finances];"Tableau"
Au cas ou
  ` ...
  :(Evenement formulaire=Sur impression pied de page)
     Si(Fin de selection([Finances]))
        VPied:="©1997 SARL Dupont" ` Définir le pied de page de la dernière page
     Sinon
        VPied:="" ` Effacer le pied de page pour toutes les autres pages
     Fin de si
Fin de cas
```

LIRE ENREGISTREMENTS MARQUES

LIRE ENREGISTREMENTS MARQUES ({laTable ;} nomEnsemble)

Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle lire les enregistrements marqués Si omis, table du formulaire courant
nomEnsemble	Chaîne	⇒	Ensemble dans lequel stocker les enregistrements marqués

Description

La commande **LIRE ENREGISTREMENTS MARQUES** stocke dans l'ensemble désigné par le paramètre *nomEnsemble* les enregistrements marqués (c'est-à-dire, les enregistrements "surlignés" par l'utilisateur dans le formulaire liste) de *laTable* passée en paramètre. Si le paramètre *laTable* est omis, la table du formulaire ou du sous-formulaire courant est utilisée.

En mode Développement ou dans le cadre de l'exécution des commandes **VISUALISER SELECTION** /**MODIFIER SELECTION**, cette commande peut être remplacée par l'appel de l'ensemble système UserSet, automatiquement maintenu par 4D. Toutefois, comme elle permet de désigner la table de laquelle récupérer les enregistrements marqués, la commande **LIRE ENREGISTREMENTS MARQUES** peut en outre gérer les sélections d'enregistrements dans les sous-formulaires inclus. En effet dans ce cas, les sélections des sous-formulaires pouvant provenir de tables différentes, l'ensemble système UserSet n'est pas géré par 4D. Pour plus d'informations sur l'ensemble UserSet, reportez-vous à la section **Ensembles**.

La commande **LIRE ENREGISTREMENTS MARQUES** peut être appelée hors du contexte d'un formulaire, cependant dans ce cas l'ensemble retourné est vide.
L'ensemble désigné par le paramètre *nomEnsemble* peut être local/client, process ou interprocess.

Note : Dans le cadre des sous-formulaires inclus, la commande **LIRE ENREGISTREMENTS MARQUES** retourne un ensemble vide si le sous-formulaire ne dispose pas de la propriété de sélection **Multilignes**. Dans ce contexte, pour connaître la ligne sélectionnée, vous devez utiliser la commande **Numero dans selection**.

Exemple

Cette méthode indique combien d'enregistrements sont sélectionnés dans le sous-formulaire affichant les enregistrements de la table [CDs] :

```
LIRE ENREGISTREMENTS MARQUES([CDs];"$highlight")
ALERTE(Chaine(Enregistrements dans ensemble("$highlight"))"+"enregistrements sélectionnés.")
EFFACER ENSEMBLE("$highlight")
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

MARQUER ENREGISTREMENTS

MARQUER ENREGISTREMENTS ({laTable }{;}{ nomEnsemble {; *}})

Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle marquer les enregistrements Si omis, table du formulaire courant
nomEnsemble	Chaîne	⇒	Ensemble d'enregistrements à marquer ou Ensemble Userset si ce paramètre est omis
*	Opérateur	⇒	Inactiver le défilement automatique de la liste

Description

La commande **MARQUER ENREGISTREMENTS** permet de "surligner" des enregistrements dans un formulaire en liste. Cette opération est identique à la sélection en mode liste, par l'utilisateur, d'enregistrement(s) à l'aide des combinaisons **Maj+clic** ou **Ctrl+clic** (Windows) ou **Commande+clic** (Mac OS). La sélection courante n'est pas modifiée.

Note : L'ensemble des enregistrements marqués est mis à jour après le redessinement des enregistrements, c'est-à-dire après la fin de l'exécution de toute la méthode d'appel — et non immédiatement après l'exécution de la commande **MARQUER ENREGISTREMENTS**.

Le paramètre *laTable* permet de désigner la table de laquelle les enregistrements doivent être "marqués". Ce paramètre permet en particulier de marquer les enregistrements des sous-formulaires inclus — n'appartenant donc pas à la table courante (cf. ci-dessous).

- Si vous passez un nom d'ensemble valide dans le paramètre *nomEnsemble*, la commande s'appliquera aux enregistrements de cet ensemble pour la *table* définie.
- Si vous omettez le paramètre *nomEnsemble*, la commande marquera les enregistrements de l'ensemble système UserSet courant. Cet ensemble est géré uniquement en mode Développement et dans le cadre de l'appel des commandes **VISUALISER SELECTION** /**MODIFIER SELECTION**). Si vous souhaitez marquer les enregistrements d'un sous-formulaire, vous devez passer un nom de table et d'ensemble. Pour plus d'informations sur l'ensemble UserSet, reportez-vous à la section **Ensembles**.

Le paramètre *, s'il est passé, provoque l'inactivation de la fonction de défilement automatique de la liste si les enregistrements marqués ne sont pas visibles. Ce mécanisme autorise la gestion personnalisée du défilement via la commande **OBJET FIXER DEFILEMENT**.

Note : Dans le cadre des sous-formulaires inclus, la commande **MARQUER ENREGISTREMENTS** ne fait rien si le sous-formulaire ne dispose pas de la propriété de sélection **Multilignes**. Dans ce contexte, pour marquer une ligne, vous devez utiliser la commande **ALLER DANS SELECTION**.

Exemple

Dans un formulaire en liste affiché par la commande **MODIFIER SELECTION**, vous souhaitez que l'utilisateur puisse effectuer des recherches, sans que la sélection courante soit modifiée. Pour cela, placez un bouton **Chercher** dans le formulaire et associez-lui la méthode suivante :

```
FIXER DESTINATION RECHERCHE(Vers ensemble;"UserSet")
CHERCHER
FIXER DESTINATION RECHERCHE(Vers sélection courante)
MARQUER ENREGISTREMENTS
```

Lorsque l'utilisateur clique sur le bouton, la boîte de dialogue standard de recherche apparaît. Une fois la recherche validée, les enregistrements trouvés sont surlignés, sans que la sélection courante ne soit modifiée.

MOBILE Renvoyer sélection

MOBILE Renvoyer sélection (laTable) -> Résultat

Paramètre	Type		Description
laTable	Table	→	Table de laquelle retourner la sélection courante
Résultat	Objet	←	Sélection compatible Wakanda

Description

La commande **MOBILE Renvoyer sélection** retourne un objet JSON contenant la sélection courante de *laTable* exprimée sous la forme d'une *entity collection* de Wakanda.

Cette commande est destinée à une utilisation dans le contexte d'une liaison 4D Mobile, établie généralement entre votre application 4D et une application Wakanda (via REST). Lorsqu'une liaison 4D Mobile est établie et que les droits d'accès appropriés ont été configurés, une application Wakanda peut exécuter une méthode projet 4D qui retourne une valeur dans le paramètre *$0*.

La commande **MOBILE Renvoyer sélection** vous permet de retourner via *$0* la sélection courante d'enregistrements de la table *laTable*, sous la forme d'un objet de type *entity collection* formaté en JSON. Cet objet est identique aux collections d'entités de Wakanda contenant une sélection d'enregistrements (i.e. d'*entités*).

N'oubliez pas que les accès 4D Mobile nécessitent des paramétrages spécifiques dans vos bases 4D :

- Le serveur Web doit être lancé,
- L'option "Activer les services 4D Mobile" doit être cochée dans les Propriétés de la base,
- Vous devez disposer d'une licence autorisant l'utilisation de 4D Mobile,
- L'option "Exposer avec le service 4D Mobile" doit être cochée pour les tables et les champs utilisés (option cochée par défaut).
- L'option "Disponible via les appels 4D Mobile" doit être cochée pour la méthode appelée (option non cochée par défaut).

A noter que vous pouvez passer toute table valide de la base dans le paramètre *laTable*, et pas nécessairement la table à laquelle la méthode projet a été associée via ses propriétés 4D Mobile. Ce paramétrage est utilisé uniquement côté Wakanda pour définir les objets sur lesquels la méthode peut être appelée.

Pour plus d'informations sur la configuration 4D Mobile, veuillez vous reporter à la documentation **4D Mobile**.

Exemple

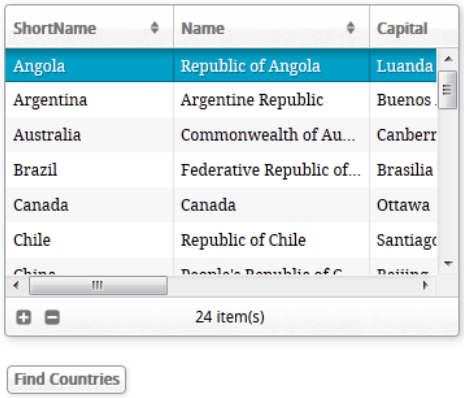
Vous souhaitez afficher la sélection courante de la table [Countries] dans une grille Wakanda, basée sur une recherche.

Vous écrivez la méthode 4D suivante :

```
   //méthode projet FindCountries
   //FindCountries( chaine ) -> objet

C_TEXTE($1)
C_OBJET($0)
CHERCHER([Countries];[Countries]ShortName=$1+"@")
$0:=MOBILE Renvoyer sélection([Countries])
```

La sélection retournée peut être utilisée directement dans Wakanda car c'est une collection valide.

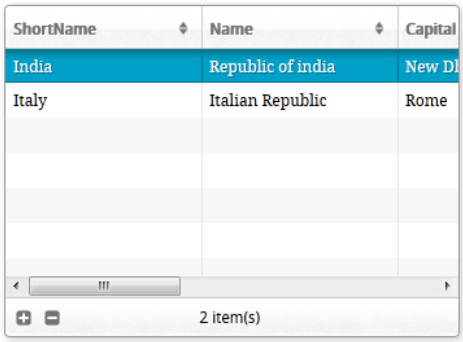
Dans le modèle du serveur Wakanda connecté à 4D via 4D Mobile, vous avez créé une page contenant une grille (*grid*) associée à la table 4D [Countries]. Par défaut, à l'exécution toutes les entités de la table 4D table sont affichées :



Le code du bouton est le suivant :

```
button1.click = function button1_click (event)     {         sources.countries.FindCountries("i", {  //on appelle la méthode 4D, "i"
est passé dans $1                 onSuccess:function(coll){  //fonction de rétroappel (asynchrone), récupère ce qui est passé dans $0
comme paramètre                     sources.countries.setEntityCollection(coll.result); //remplace l'entity collection courante
                          // avec celle reçue dans l'objet coll.result                 }             });            };
```

En résultat, la grille est mise à jour :


ShortName
Name
Capital
India
Republic of india
New Dl
Italy
Italian Republic
Rome
2 item(s)
Find Countries

MODIFIER SELECTION

MODIFIER SELECTION ({laTable}{; modeSélection}{; saisieListe}{; *}{; *})

Paramètre	Type		Description
laTable	Table	⇒	Table à afficher et modifier ou Table par défaut si ce paramètre est omis
modeSélection	Entier long	⇒	Mode de sélection
saisieListe	Booléen	⇒	Autoriser saisie en liste
*		⇒	Utiliser formulaire sortie pour un seul enregistrement et cacher les barres de défilement dans le formulaire entrée
*		⇒	Afficher les barres de défilement dans le formulaire entrée (= annuler le second effet du premier paramètre *)

Description

La commande **MODIFIER SELECTION** est quasiment identique à la commande **VISUALISER SELECTION**. Reportez-vous à la commande **VISUALISER SELECTION** pour une description détaillée.
Les seules différences entre ces deux commandes sont les suivantes :

1. **VISUALISER SELECTION** et **MODIFIER SELECTION** provoquent l'affichage des enregistrements de la sélection courante de *laTable* dans le formulaire sortie courant, ou dans le formulaire entrée lorsque vous double-cliquez sur un enregistrement. Avec **MODIFIER SELECTION**, vous pouvez en plus modifier les champs de l'enregistrement dans le formulaire entrée lorsque vous double-cliquez dessus (s'il n'est pas déjà chargé par un autre utilisateur/process) ou en mode “Saisie en liste” (s'il est autorisé).

2. **VISUALISER SELECTION** charge les enregistrements en mode Lecture seulement dans le process courant, ce qui signifie qu'ils ne sont pas verrouillés en écriture pour les autres process. **MODIFIER SELECTION** place tous les enregistrements de la sélection en mode Lecture-écriture, ce qui signifie qu'ils sont automatiquement verrouillés en écriture pour les autres process. **MODIFIER SELECTION** libère les enregistrements lorsque son exécution est terminée.

Numero dans selection

Numero dans selection {(laTable)} -> Résultat

Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle retourner le numéro de l'enregistrement courant dans la sélection
Résultat	Entier long	⇐	Numéro dans la sélection

Description

Numero dans selection retourne la position de l'enregistrement courant dans la sélection courante de *laTable*.

Si la sélection est non vide et si l'enregistrement courant en fait partie, **Numero dans selection** retourne une valeur comprise entre *1* et **Enregistrements trouves**. Si la sélection est vide ou s'il n'y a pas d'enregistrement courant, **Numero dans selection** retourne 0.

Le numéro de l'enregistrement dans la sélection est différent du numéro retourné par **Numero enregistrement** (**Numero enregistrement** retourne le numéro physique de l'enregistrement dans la table). Le numéro de l'enregistrement dans la sélection dépend de la sélection courante.

Pour plus de précisions sur les numéros d'enregistrements, reportez-vous à la section **A propos des numéros d'enregistrements**.

Exemple

L'exemple suivant stocke le numéro de l'enregistrement courant de la sélection dans une variable :

```
   ` Obtenir le numéro de l'enregistrement dans la sélection
NumEnrCourant:=Numero dans selection([Personnes])
```

Numero de ligne affichee

Numero de ligne affichee -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Numéro de ligne en cours d'affichage

Description

La commande **Numero de ligne affichee** fonctionne uniquement dans le contexte de l'événement formulaire Sur affichage corps. Elle retourne le numéro de la ligne en cours de traitement durant l'affichage à l'écran d'une liste d'enregistrements ou des lignes d'une list box. Si **Numero de ligne affichee** est appelée en-dehors de l'affichage d'une liste ou d'une listbox, elle retourne 0.

Dans le cas d'une liste d'enregistrements, lorsque la ligne affichée n'est pas vide (c'est-à-dire lorsqu'elle est associée à un enregistrement), la valeur retournée par **Numero de ligne affichee** est identique à celle retournée par **Numero dans selection**.

Comme **Numero dans selection**, **Numero de ligne affichee** débute à 1. Cette commande est utile lorsque vous souhaitez appliquer un traitement à chaque ligne d'un formulaire liste ou d'une list box affiché(e) à l'écran, y compris aux lignes vides.

Exemple

Cet exemple permet d'appliquer une couleur alternée à un formulaire liste affiché à l'écran, même pour les lignes sans enregistrement :

```
  `Méthode du formulaire liste
Si(Evenement formulaire=Sur affichage corps)
   Si(Numero de ligne affichee% 2=0)
  `Noir sur blanc pour le texte des lignes paires
      OBJET FIXER COULEURS RVB([Table 1]Champ1;-1;0x00FFFFFF)
   Sinon
  `Noir sur bleu pâle pour le texte des lignes impaires
      OBJET FIXER COULEURS RVB([Table 1]Champ1;-1;0x00E0E0FF)
   Fin de si
Fin de si
```

REDUIRE SELECTION

REDUIRE SELECTION ({laTable ;} nombre)

Paramètre	Type		Description
laTable	Table	➡	Table de laquelle réduire la sélection ou Table par défaut si ce paramètre est omis
nombre	Entier long	➡	Nombre d'enregistrements à conserver

Description

La commande **REDUIRE SELECTION** crée une nouvelle sélection d'enregistrements pour *laTable*. La commande réduit la sélection de *laTable* aux *nombre* premiers enregistrements. **REDUIRE SELECTION** s'applique à la sélection courante de *laTable* pour le process courant. Le premier enregistrement de la nouvelle sélection courante devient l'enregistrement courant.

Note : Si l'instruction **REDUIRE SELECTION**(laTable;0) est exécutée, il n'y a plus de sélection ni d'enregistrement courants dans la table.

Exemple

L'exemple suivant établit des statistiques pour une compétition mondiale parmi les revendeurs dans plus de 20 pays. Pour chaque pays, les trois meilleurs revendeurs qui ont vendu plus de 50 000 Euros de produits font partie des 100 meilleurs revendeurs dans le monde et sont recompensés. Avec peu de lignes de code, cette requête complexe peut être effectuée en utilisant des recherches indexées :

```
ENSEMBLE VIDE([Revendeurs];"Gagnants") ` Créer un ensemble vide
SCAN INDEX([Revendeurs]Montant;100;<) ` Chercher à la fin de l'index
NOMMER ENSEMBLE([Revendeurs];"100 Meilleurs Revendeurs")
 ` Placer les enregistrements sélectionnés dans un ensemble
Boucle($Pays;1;Enregistrements dans table([Pays])) ` pour chaque pays
 ` Chercher les revendeurs dans ce pays
  CHERCHER([Revendeurs];[Revendeurs]Pays=NomPays;*) ` ...qui ont vendu plus de 50000
  CHERCHER(&;[Revendeurs];[Revendeurs]Montant vendu>=50000)
  NOMMER ENSEMBLE([Revendeurs];"GagnantsRevendeurs") ` Les placer dans un ensemble
 ` Ils doivent être placés dans le groupe des 100 meilleurs revendeurs
  INTERSECTION("GagnantsRevendeurs";"100 Meilleurs Revendeurs";"GagnantsRevendeurs")
  UTILISER ENSEMBLE("GagnantsRevendeurs") ` Gagnants potentiels pour le pays
 ` Trier les résultats en ordre décroissant
  TRIER([Revendeurs];[Revendeurs]Montant vendu;<)
  REDUIRE SELECTION([Revendeurs];3) ` Garder les trois meilleurs
  NOMMER ENSEMBLE([Revendeurs];"GagnantsRevendeurs") ` Les gagnants pour le pays
 ` Les placer dans un ensemble des gagnants mondiaux
  REUNION("GagnantsRevendeurs";"LesGagnants";"LesGagnants")
Fin de boucle
EFFACER ENSEMBLE("100 Meilleurs Revendeurs") ` Nous n'avons plus besoin de cet ensemble
EFFACER ENSEMBLE("GagnantsRevendeurs") ` Nous n'avons plus besoin de cet ensemble
UTILISER ENSEMBLE("LesGagnants") ` Voici les gagnants
EFFACER ENSEMBLE("LesGagnants") ` Nous n'avons plus besoin de cet ensemble
FORMULAIRE SORTIE([Revendeurs];"Lettre des gagnants") ` Sélectionner la lettre
IMPRIMER SELECTION([Revendeurs]) ` Imprimer les lettres
```

SCAN INDEX

SCAN INDEX (leChamp ; nombre {; > ou <})

Paramètre	Type		Description
leChamp	Champ	⇒	Champ indexé avec lequel "scanner" les enregistrements
nombre	Entier long	⇒	Nombre d'enregistrements à retourner
> ou <	Opérateur	⇒	> à partir du début de l'index < à partir de la fin de l'index

Description

La commande **SCAN INDEX** retourne une sélection de *nombre* enregistrements de la table du champ *leChamp*. Cette commande est extrêmement rapide car elle utilise l'index.
Si vous passez <, **SCAN INDEX** retourne *nombre* enregistrements à partir de la fin de l'index (valeurs supérieures). Si vous passez >, **SCAN INDEX** retourne *nombre* enregistrements à partir du début de l'index (valeurs inférieures). Si vous ne passez pas le dernier paramètre, la commande retourne *nombre* enregistrements à partir du début de l'index (équivaut à passer >).

Note : La sélection obtenue n'est pas triée.

SCAN INDEX fonctionne uniquement avec des champs indexés. Cette commande modifie la sélection courante de la table pour le process courant et charge le premier enregistrement de la sélection en tant qu'enregistrement courant.

Si vous spécifiez un nombre d'enregistrements supérieur au nombre d'enregistrements présents dans la table, **SCAN INDEX** retourne tous les enregistrements.

Note : Cette commande ne prend pas en charge les champs de type Objet.

Exemple

L'exemple suivant envoie des lettres aux 50 plus mauvais clients puis aux 50 meilleurs clients :

```
SCAN INDEX([Clients]TotalDû;50;<) ` Obtenir la liste des 50 plus mauvais clients
TRIER([Clients]CodePostal;>) ` Trier par code postal
FORM FIXER SORTIE([Clients];"Menace")
IMPRIMER SELECTION([Clients]) ` Imprimer les lettres
SCAN INDEX([Clients]TotalDû;50;>) ` Obtenir la liste des 50 meilleurs clients
TRIER([Clients]CodePostal;>) ` Trier par code postal
FORM FIXER SORTIE([Clients];"Remerciement")
IMPRIMER SELECTION([Clients]) ` Imprimer les lettres
```

SUPPRIMER SELECTION

SUPPRIMER SELECTION {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle supprimer la sélection courante ou Table par défaut si ce paramètre est omis

Description

La commande **SUPPRIMER SELECTION** supprime la sélection courante d'enregistrements de *laTable*. Si la sélection courante est vide, **SUPPRIMER SELECTION** ne fait rien. Après la suppression des enregistrements, la sélection courante est vide. Les enregistrements supprimés pendant une transaction sont verrouillés pour les autres utilisateurs et/ou process jusqu'à ce que la transaction soit validée ou annulée.

Attention : La suppression d'une sélection d'enregistrements est une opération définitive. Elle ne peut être annulée par la suite.

Désélectionner l'option **Enregistrement(s) définitivement supprimé(s)** dans l'Inspecteur des tables vous permet d'augmenter la vitesse des suppressions lors de l'utilisation de **SUPPRIMER SELECTION** (cf. paragraphe **Enregistrement(s) définitivement supprimé(s)** dans le manuel *Mode Développement*).

Exemple 1

L'exemple suivant affiche tous les enregistrements de la table [Personnes] et permet à l'utilisateur de sélectionner ceux qu'il souhaite effacer. L'exemple est en deux parties. La première est la méthode affichant les enregistrements. La seconde est la méthode objet d'un bouton 'Supprimer'. Voici la première méthode :

```
TOUT SELECTIONNER([Personnes]) ` Sélection de tous les enregistrements
FORM FIXER SORTIE([Personnes];"FormSortie") ` Définition du formulaire listant les enregistrements
VISUALISER SELECTION([Personnes]) ` Affichage de tous les enregistrements
```

Voici la méthode objet du bouton Supprimer, apparaissant dans le pied de page du formulaire sortie. La méthode utilise les enregistrements sélectionnés par l'utilisateur (l'ensemble système *UserSet*) pour effacer la sélection (notez que si l'utilisateur ne sélectionne aucun enregistrement, **SUPPRIMER SELECTION** ne fait rien) :

```
 ` Demander confirmation que l'utilisateur veut réellement supprimer les enregistrements
CONFIRMER("Vous avez sélectionné "+Chaine(Enregistrements dans ensemble("UserSet"))+
" enregistrements à supprimer."+Caractere(13)+"Cliquez sur OK pour confirmer l'opération.")
Si(OK=1)
   UTILISER ENSEMBLE("UserSet") ` Utiliser l'ensemble défini par l'utilisateur
   SUPPRIMER SELECTION([Personnes]) ` Supprimer la sélection d'enregistrements
Fin de si
TOUT SELECTIONNER([Personnes]) ` Sélection de tous les enregistrements
```

Exemple 2

Lorsqu'un **SUPPRIMER SELECTION** rencontre un enregistrement verrouillé, celui-ci n'est pas supprimé. Tous les enregistrements verrouillés sont placés dans un ensemble système nommé *LockedSet*. Après l'exécution de **SUPPRIMER SELECTION**, vous pouvez tester cet ensemble afin de vérifier si des enregistrements étaient verrouillés. La boucle suivante s'exécutera jusqu'à ce que tous les enregistrements aient été supprimés.

```
Repeter ` Répéter pour chaque enregistrement verrouillé
   SUPPRIMER SELECTION([CetteTable])
   Si(Enregistrements dans ensemble("LockedSet")#0) ` Si des enregistrements sont verrouillés
      UTILISER ENSEMBLE("LockedSet") ` Sélectionner les enregistrements verrouillés
   Fin de si
Jusque(Enregistrements dans ensemble("LockedSet")=0) ` Jusqu'à ce qu'il n'y en ait plus
```

TOUT SELECTIONNER

TOUT SELECTIONNER {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle vous voulez sélectionner tous les enregistrements ou Table par défaut si ce paramètre est omis

Description

La commande **TOUT SELECTIONNER** sélectionne tous les enregistrements de *laTable* pour le process courant. **TOUT SELECTIONNER** fait du premier enregistrement de la sélection l'enregistrement courant et le charge en mémoire. **TOUT SELECTIONNER** retourne les enregistrements dans l'ordre par défaut, qui est l'ordre dans lequel ils ont été stockés sur le disque.

Exemple

L'exemple suivant affiche tous les enregistrements de la table [Personnes] :

```
TOUT SELECTIONNER([Personnes]) ` Sélection de tous les enregistrements de la table
VISUALISER SELECTION([Personnes]) `Affichage des enregistrements dans le formulaire sortie
```

VIDER TABLE

VIDER TABLE {(laTable)}

Paramètre	Type		Description
laTable	Table	⇨	Table de laquelle vous voulez supprimer tous les enregistrements ou Table par défaut si ce paramètre est omis

Description

La commande **VIDER TABLE** supprime tous les enregistrements de *laTable* de façon très rapide. Après l'appel de la commande, il n'y a plus de sélection courante ni d'enregistrement courant.

L'effet de cette commande est semblable à celui d'une séquence **TOUT SELECTIONNER** / **SUPPRIMER SELECTION**, toutefois son fonctionnement diffère sur les points suivants :

- Le trigger éventuel n'est pas appelé.
- L'intégrité référentielle des données n'est pas contrôlée.
- Aucune transaction ne doit être en cours dans le process exécutant **VIDER TABLE**. Si c'est le cas, la commande ne fait rien et la variable système OK prend la valeur 0.
- Si un enregistrement au moins est verrouillé par un autre process, la commande échoue : une erreur est générée et la variable OK prend la valeur 0. L'ensemble système LockedSet n'est pas créé.
- Si *laTable* est déjà vide, **VIDER TABLE** ne fait rien et fixe la variable OK à 1.
- Si *laTable* est en lecture seule, **VIDER TABLE** ne fait rien et fixe la variable OK à 0.
- L'opération est enregistrée dans le fichier d'historique s'il est présent.

La commande **VIDER TABLE** est donc à manier avec précaution mais est très efficace pour, par exemple, supprimer rapidement des données temporaires.

Note : Le concept et le fonctionnement de cette commande sont proches de ceux de la commande TRUNCATE (TABLE) du SQL.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

VISUALISER SELECTION

VISUALISER SELECTION ({laTable}{; modeSélection}{; saisieListe}{; *}{; *})

Paramètre	Type		Description
laTable	Table	➡	Table à laquelle appartient la sélection ou Table par défaut si ce paramètre est omis
modeSélection	Entier long	➡	Mode de sélection
saisieListe	Booléen	➡	Autoriser saisie en liste
*		➡	Utiliser le formulaire sortie en cas de sélection d'un seul enregistrement et masquer les barres de défilement dans le formulaire entrée
*		➡	Afficher les barres de défilement dans le formulaire entrée (= annuler le second effet du premier paramètre *)

Description

VISUALISER SELECTION affiche, pour le process en cours, la sélection courante de *laTable* dans le formulaire sortie courant. Les enregistrements sont affichés sous la forme d'une liste que l'on peut faire défiler, semblable à celle du mode Développement. Lorsque l'utilisateur double-clique sur un enregistrement, par défaut celui-ci s'affiche dans le formulaire entrée courant. La liste est placée dans la fenêtre de premier plan.

Si vous souhaitez afficher une sélection et pouvoir également modifier un enregistrement dans le formulaire entrée courant une fois que vous avez double-cliqué dessus (comme vous le faites dans la fenêtre du mode Développement) ou via le mode “Saisie en liste”, utilisez **MODIFIER SELECTION** au lieu de **VISUALISER SELECTION**. Toutes les explications suivantes s'appliquent à ces deux commandes, hormis la possibilité de modifier des enregistrements.

Après qu'un **VISUALISER SELECTION** ait été exécuté, il n'y a plus d'enregistrement courant. Vous devez utiliser une commande telle que **DEBUT SELECTION** ou **ALLER A DERNIER ENREGISTREMENT** pour en récupérer un.

Le paramètre *modeSélection* vous permet de définir les possibilités de sélection d’enregistrements dans la liste à l'aide de la souris. Vous pouvez passer dans ce paramètre une des constantes suivantes du thème “**Paramètres de formulaire**” :

Constante	Type	Valeur	Comment
Pas de sélection	Entier long	0	Il n'est pas possible de sélectionner un enregistrement dans la liste
Sélection multiple	Entier long	2	L’utilisateur peut sélectionner plusieurs enregistrements. Pour sélectionner des enregistrements contigus, il suffit de cliquer sur le premier enregistrement à sélectionner puis d'appuyer sur la touche **Majuscule** avant de cliquer sur le dernier. Pour sélectionner des enregistrements non adjacents, il suffit de cliquer sur chaque enregistrement en maintenant enfoncée la touche **Ctrl** (sous Windows) ou **Commande** (sous Mac OS).
Sélection unique	Entier long	1	Seule la sélection d’un enregistrement à la fois est autorisée

Si vous ne passez pas le paramètre *modeSélection*, par défaut le mode “Sélection multiple” est utilisé.

Le paramètre *saisieListe* vous permet d’autoriser le mode “Saisie en liste” dans la liste affichée. Ce mode permet à l'utilisateur de sélectionner et de modifier directement les valeurs des enregistrements dans le formulaire sortie. Passez **Vrai** pour autoriser ce mode ou **Faux** pour ne pas l’autoriser. Par défaut, si vous ne passez pas le paramètre *saisieListe*, le mode “Saisie en liste” n’est pas autorisé.
A noter qu'avec la commande **VISUALISER SELECTION**, ce paramètre permet uniquement la sélection de valeurs dans la liste et non leur modification. En effet, la commande **VISUALISER SELECTION** charge les enregistrements de la sélection courante en Lecture seulement dans le process en cours. Seule la commande **MODIFIER SELECTION** permet effectivement la saisie de valeurs.

Note : La commande **OBJET FIXER SAISISSABLE** permet d’activer ou de désactiver le mode Saisie en liste à la volée.

Lorsque la sélection ne contient qu'un enregistrement, et que le premier paramètre optionnel * n'est pas passé, l'enregistrement s'affichera directement dans le formulaire entrée. Si le premier paramètre optionnel * est spécifié, l'enregistrement unique sera affiché dans le formulaire sortie. Si le premier paramètre optionnel * est spécifié et que l'utilisateur affiche l'enregistrement dans le formulaire entrée en double-cliquant dessus, les barres de défilement du formulaire seront masquées. Pour annuler ce second effet du premier paramètre optionnel *, passez le second paramètre optionnel *.

Vous pouvez placer des boutons personnalisés dans la zone d'en-tête ou de pied de page du formulaire sortie pour terminer l'exécution de la commande **VISUALISER SELECTION**. Vous pouvez utiliser des boutons automatiques **Valider** ou **Annuler** permettant de sortir de la liste ou utiliser une méthode objet qui appelle les commandes **VALIDER** ou **NE PAS VALIDER**. Lorsqu’un formulaire sortie appelé par la commande **VISUALISER SELECTION** est dépourvu de boutons, seule la touche **Echap** (Windows) ou **Esc** (Mac OS) permet de quitter la liste.

Pendant et après l'exécution d'un **VISUALISER SELECTION**, les enregistrements sélectionnés par l'utilisateur sont conservés dans un ensemble système nommé *UserSet*. Après l'exécution de la commande, l'ensemble *UserSet* est accessible pendant un **VISUALISER SELECTION** aux méthodes objet de boutons, aux méthodes appelées par des commandes de menu, ainsi que pour la méthode projet qui avait appelé **VISUALISER SELECTION**.

Exemple 1

L'exemple suivant sélectionne tous les enregistrements de la table [Personnes]. La commande **VISUALISER SELECTION** est alors utilisée pour afficher les enregistrements et permettre à l'utilisateur de désigner ceux qu'il souhaite imprimer. Enfin, les enregistrements sélectionnés sont récupérés à l'aide de la commande **UTILISER ENSEMBLE** et imprimés avec **IMPRIMER SELECTION** :

```
TOUT SELECTIONNER([Personnes]) ` Sélection de tous les enregistrements
VISUALISER SELECTION([Personnes];*) ` Affichage des enregistrements
UTILISER ENSEMBLE("UserSet") ` Utiliser uniquement les enregistrements sélectionnés par l'utilisateur
IMPRIMER SELECTION([Personnes]) ` Imprimer les enregistrements sélectionnés
```

Exemple 2

Reportez-vous à l'exemple n°6 de la commande **Evenement formulaire** ; il indique tous les tests que vous pourrez avoir besoin d'effectuer pour surveiller la totalité des événements intervenant pendant l'exécution de la commande **VISUALISER SELECTION**.

Exemple 3

Pour reproduire, par exemple, les fonctionnalités apportées par le menu **Enregistrements** du mode Développement lorsque vous utilisez **MODIFIER**

SELECTION ou **VISUALISER SELECTION** en mode Application, procédez de la manière suivante :

I. Dans le mode Développement, créez une barre de menus comportant les menus qui vous intéressent (par exemple Tout montrer, Recherche et Trier).
II. Associez cette barre de menus (à l'aide du menu “Barre de menus associée” dans la boîte de dialogue des propriétés du formulaire) au formulaire sortie utilisé avec les commandes **VISUALISER SELECTION** ou **MODIFIER SELECTION**.
III. Associez les méthodes projet suivantes à vos commandes de menu :

```
  ` M_TOUT_MONTRER (associée à la commande de menu Tout montrer)
$vpCourTable:=Table du formulaire courant
TOUT SELECTIONNER($vpCourTable->)
```

```
  ` M_Recherche (associée à la commande de menu Recherche)
$vpCourTable:=Table du formulaire courant
CHERCHER($vpCourTable->)

  ` M_TRIER (associée à la commande de menu Trier)
$vpCourTable:=Table du formulaire courant
TRIER($vpCourTable->)
```

Vous pouvez aussi utiliser d'autres commandes telles que **IMPRIMER SELECTION**, **QR ETAT**, etc. , afin de reproduire les commandes de menu "standard" à chaque fois que vous affichez ou modifiez une sélection en mode Application. Grâce à la commande **Table du formulaire courant**, ces méthodes sont génériques et les barres de menus auxquelles elles sont associées peuvent être rattachées à tout formulaire de sortie ou à toute table.

Sélections Temporaires

Présentation des Sélections Temporaires

Les sélections temporaires vous permettent de manipuler plusieurs sélections à la fois. Une sélection temporaire est une liste ordonnée d'enregistrements pour une table dans un process. Cette liste ordonnée d'enregistrements peut avoir un nom et est conservée en mémoire. Les sélections temporaires vous fournissent un moyen facile de garder en mémoire l'ordre et l'enregistrement courant de la sélection.

Les commandes suivantes vous permettent de travailler avec les sélections temporaires :

- **COPIER SELECTION**
- **DEPLACER SELECTION**
- **UTILISER SELECTION**
- **EFFACER SELECTION**
- **CREER SELECTION SUR TABLEAU**

Les sélections temporaires sont créées par les commandes **COPIER SELECTION**, **DEPLACER SELECTION** et **CREER SELECTION SUR TABLEAU**. Les sélections temporaires sont généralement utilisées pour travailler avec une ou plusieurs sélections, effectuer une sauvegarde puis retrouver une sélection ordonnée. Il peut y avoir plusieurs sélections temporaires pour chaque table dans un process. Pour réutiliser une sélection temporaire en tant que sélection courante, appelez **UTILISER SELECTION**. Lorsque vous en avez terminé avec une sélection temporaire, utilisez **EFFACER SELECTION**.

Note : La combinaison de l'instruction **FIXER DESTINATION RECHERCHE(Vers sélection temporaire;selectiontemp)** et d'une commande de recherche (par exemple **CHERCHER**) permet également de créer une sélection temporaire. Reportez-vous à la description de la commande **FIXER DESTINATION RECHERCHE**.

Les sélections temporaires peuvent avoir une portée (une aire d'action) locale, process ou interprocess.

Une sélection temporaire est locale lorsque son nom est précédé du symbole $. Elle est process lorsque son nom n'est précédé d'aucun symbole. Elle est interprocess lorsque son nom est précédé des symboles (<>) — le signe “inférieur à” suivi du symbole “supérieur à”.

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. Sous Mac OS, vous pouvez aussi utiliser le symbole "diamant" (Option + v sur un clavier français).

La portée d'une sélection temporaire interprocess est identique à celle d'une variable interprocess. On peut accéder à une sélection temporaire interprocess à partir de n'importe quel process.
Avec 4D en mode distant et 4D Server, une sélection temporaire interprocess n'est accessible que pour les process du client qui l'a créée. Une sélection temporaire interprocess n'est pas accessible aux autres clients.

Une sélection temporaire process n'est disponible que dans le process où elle a été créée et sur le serveur.

Une sélection temporaire locale est définie pour le process qui l'a créée et n'est pas visible sur le serveur.

Attention : Créer une sélection temporaire nécessite l'accès à la sélection de la table. Comme les sélections sont conservées sur le serveur et qu'un process local n'a pas accès à 4D Server, ne cherchez pas à utiliser des sélections temporaires dans un process local.

Visibilité des sélections temporaires

Le tableau suivant indique les principes de visibilité des sélections temporaires en fonction de leur portée et de leur lieu de création :

	Process client	Autres process du client	Autres clients	Process serveur	Autres process du serveur
Création dans un process client					
$test	x				
test	x			x (Trigger)	
<>test	x	x			
Création dans un process serveur					
$test				x	
test				x	
<>test				x	x

Sélections temporaires et ensembles

Voici les différences majeures entre les ensembles et les sélections temporaires :

- Une sélection temporaire est une liste ordonnée d'enregistrements, ce que n'est pas un ensemble.
- Les ensembles sont économes en mémoire car il n'ont besoin que d'un bit par enregistrement de la table. Les sélections temporaires ont besoin de 4 octets pour chaque enregistrement dans la sélection.
- A la différence des ensembles, les sélections temporaires ne peuvent pas être sauvegardées sur disque.
- Alors que les opérations standard **INTERSECTION**, **REUNION** et **DIFFERENCE** sont possibles pour les ensembles, les sélections temporaires ne peuvent être combinées avec d'autres sélections temporaires.

Les similitudes entre les sélections temporaires et les ensembles sont les suivantes :

- Comme un ensemble, une sélection temporaire existe en mémoire.
- Une sélection temporaire et un ensemble stockent des références aux enregistrements. Si des enregistrements sont modifiés ou détruits, la sélection temporaire ou l'ensemble peuvent n'être plus valides.
- Comme un ensemble, une sélection temporaire repère l'enregistrement courant au moment où elle est créée.

COPIER SELECTION

COPIER SELECTION ({laTable ;} nom)

Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle il faut copier la sélection ou Table par défaut si ce paramètre est omis
nom	Chaîne	⇒	Nom de la sélection temporaire à créer

Description

COPIER SELECTION copie la sélection courante de *laTable* dans une sélection temporaire *nom*. La table par défaut du process courant est utilisée si le paramètre optionnel *laTable* n'est pas spécifié. La sélection temporaire *nom* contient une copie de la sélection. La sélection courante et l'enregistrement courant de *laTable* pour le process courant ne sont pas modifiés.

Une sélection temporaire ne contient pas les enregistrements, mais une liste triée des références aux enregistrements. Chaque référence à un enregistrement prend 4 octets en mémoire. Ceci signifie que lorsqu'une sélection est copiée à l'aide de la commande **COPIER SELECTION**, la mémoire requise est 4 octets multipliés par le nombre d'enregistrements dans la sélection. Comme les sélections temporaires restent en mémoire, il vous faut assez de mémoire pour la sélection temporaire ainsi que la sélection courante de la table pour le process.

4D Server : La sélection temporaire *nom* ainsi que la sélection courante sont logées dans la mémoire du poste serveur. En conséquence, assurez-vous que le serveur dispose de suffisamment de mémoire.

Utilisez la commande **EFFACER SELECTION** pour libérer la mémoire utilisée par *nom*.

Exemple

L'exemple suivant permet de vérifier s'il y a des factures impayées dans la table [Personnes]. La sélection est triée puis sauvegardée. Nous cherchons toutes les factures qui n'ont pas été payées. Ensuite, nous réutilisons la sélection et effaçons la sélection temporaire en mémoire :

```
TOUT SELECTIONNER([Personnes])
 `Permettre à l'utilisateur de trier la sélection
TRIER([Personnes])
 ` Stocker la sélection dans une sélection temporaire
COPIER SELECTION([Personnes];"TriéeUtilisateur")
 ` Rechercher les factures impayées
CHERCHER([Personnes];[Personnes]FactureDue=Vrai)
 ` Si un enregistrement a été trouvé
Si(Enregistrements trouves([Personnes])>0)
 ` Informer l'utilisateur
   ALERTE("Oui, quelques factures n'ont pas été réglées.")
Fin de si
 ` Réutiliser la sélection temporaire triée
UTILISER SELECTION("TriéeUtilisateur")
 ` Effacer la sélection de la mémoire
EFFACER SELECTION("TriéeUtilisateur")
```

DEPLACER SELECTION

DEPLACER SELECTION ({laTable ;} nom)

Paramètre	Type		Description
laTable	Table	⇒	Table de la sélection ou Table par défaut si ce paramètre est omis
nom	Chaîne	⇒	Nom de la sélection temporaire à créer

Description

DEPLACER SELECTION crée la sélection temporaire *nom* et y place la sélection courante de *laTable*. A la différence de **COPIER SELECTION**, cette commande ne copie pas la sélection, mais la déplace.

Après l'exécution de cette commande, la sélection courante de *laTable* dans le process courant est vide. En conséquence, **DEPLACER SELECTION** ne doit pas être utilisée lorsqu'un enregistrement est en cours de modification.

En termes d'utilisation de la mémoire, **DEPLACER SELECTION** est plus économique que **COPIER SELECTION**. En effet, **COPIER SELECTION** utilise 4 octets de mémoire pour chaque enregistrement de la sélection. Avec **DEPLACER SELECTION**, seule la référence à la sélection est déplacée.

Exemple

La méthode suivante vide la sélection courante de la table *[Clients]* :

```
DEPLACER SELECTION([Clients];"AEffacer")
EFFACER SELECTION("AEffacer")
```

EFFACER SELECTION

EFFACER SELECTION (nom)

Paramètre	Type		Description
nom	Chaîne	⇒	Nom de la sélection temporaire à effacer

Description

EFFACER SELECTION efface *nom* de la mémoire et donc libère la mémoire qu'elle utilisait. **EFFACER SELECTION** n'affecte pas les tables, sélections courantes ou enregistrements. Comme les sélections temporaires utilisent de la mémoire, il est conseillé de les effacer si vous n'en avez plus besoin.

Si *nom* a été créée par la commande **DEPLACER SELECTION** puis traitée à l'aide de la commande **UTILISER SELECTION**, elle n'existe plus en mémoire. Dans ce cas, vous n'avez pas besoin d'utiliser **EFFACER SELECTION**.

UTILISER SELECTION

UTILISER SELECTION (nom)

Paramètre	Type		Description
nom	Chaîne	⇨	Nom de la sélection temporaire à utiliser

Description

UTILISER SELECTION désigne la sélection temporaire *nom* comme sélection courante pour la table à laquelle elle appartient.

Lorsque vous créez une sélection temporaire, l'enregistrement courant est aussi stocké par la sélection temporaire. **UTILISER SELECTION** récupère la position de cet enregistrement et en fait l'enregistrement courant. L'enregistrement courant est alors chargé. S'il a été modifié après la création de la sélection temporaire *nom*, il doit être sauvegardé avant que la commande **UTILISER SELECTION** soit appelée, afin de ne pas perdre les informations modifiées.

- Si *nom* a été créée par la commande **COPIER SELECTION**, la sélection temporaire est utilisée comme sélection courante de la table à laquelle elle appartient. La sélection temporaire *nom* existe en mémoire jusqu'à ce qu'elle soit effacée. Pour récupérer l'espace mémoire occupé par *nom*, appelez la commande **EFFACER SELECTION**.
- Si *nom* a été créée par la commande **DEPLACER SELECTION**, elle est utilisée comme sélection courante de la table à laquelle elle appartient et *nom* n'existe plus en mémoire.

N'oubliez pas qu'une sélection temporaire est la représentation d'une sélection courante à un instant donné. Si les enregistrements que la sélection temporaire représente sont modifiés, celle-ci devient obsolète. En conséquence, une sélection temporaire doit représenter une sélection d'enregistrements dont le contenu est relativement stable.
Différents événements peuvent rendre une sélection temporaire obsolète : la modification ou la suppression d'un enregistrement appartenant à la sélection temporaire ou la modification des critères de création de la sélection temporaire.

Serveur Web

- Présentation du serveur Web
- Mise en route du serveur Web et gestion des connexions
- Prise en charge d'IP v6
- Sécurité des connexions
- Méthode base Sur authentification Web
- Méthode base Sur connexion Web
- Méthode base Sur fermeture process Web
- Gestion des sessions Web
- Pages semi-dynamiques
- URLs et actions de formulaires
- Traiter les données reçues
- Paramétrages du serveur Web
- Utiliser des process Web préemptifs
- Informations sur le site Web
- Utiliser le protocole TLS
- Support de XML et de WML
- WEB ARRETER SERVEUR
- WEB Connexion securisee
- WEB DEMARRER SERVEUR
- WEB ENVOYER BLOB
- WEB ENVOYER DONNEES
- WEB ENVOYER FICHIER
- WEB ENVOYER REDIRECTION HTTP
- WEB ENVOYER TEXTE
- WEB FERMER SESSION
- WEB FIXER ENTETE HTTP
- WEB FIXER OPTION
- WEB FIXER PAGE ACCUEIL
- WEB FIXER RACINE
- WEB LIRE CORPS HTTP
- WEB LIRE ENTETE HTTP
- WEB LIRE EXPIRATION SESSION
- WEB Lire ID session courante
- WEB Lire nombre parties corps
- WEB Lire nombre process session
- WEB LIRE OPTION
- WEB LIRE PARTIE CORPS
- WEB LIRE STATISTIQUES
- WEB LIRE VARIABLES
- WEB Serveur est démarré
- WEB Valider digest
- *_o_Contexte Web*
- *_o_FIXER EXECUTABLE CGI*
- *_o_FIXER LIMITES AFFICHAGE WEB*
- *_o_FIXER TEMPORISATION WEB*

Présentation du serveur Web

4D en mode local, 4D en mode distant et 4D Server contiennent un serveur Web qui vous permet de publier des bases 4D ou tout type de page HTML sur le Web. Les principales caractéristiques du moteur du serveur Web de 4D sont les suivantes :

- **Simplicité de publication**

Vous pouvez à tout moment lancer ou stopper la publication de la base sur le Web. Pour cela, il suffit de choisir une commande de menu ou d'exécuter une commande du langage.

- **Méthodes base dédiées**

La **Méthode base Sur authentification Web** et la **Méthode base Sur connexion Web** constituent les points d'entrée des requêtes dans le serveur Web ; elles peuvent être utilisées pour évaluer et acheminer tout type de requête.

- **Utilisation de balises et d'URLs spéciaux**

Le serveur Web de 4D propose de nombreux mécanismes permettant d'interagir avec les actions des utilisateurs, notamment :
- des balises spéciales peuvent être incluses dans les pages Web afin de provoquer des traitements par le serveur Web au moment de leur envoi aux navigateurs.
- des URLs spéciaux permettent d'appeler 4D afin d'exécuter toute action.
- ces URLs peuvent également être utilisés comme actions de formulaire pour déclencher des traitements lorsque l'utilisateur poste des formulaires HTML.

- **Gestion des sessions utilisateur**

Le serveur Web de 4D inclut des automatismes complets permettant de gérer facilement les sessions Web (sessions utilisateur) basées sur les cookies.

- **Sécurité des accès**

Des options de configuration automatiques vous permettent d'accorder des autorisations d'accès spécifiques aux navigateurs Web ou d'utiliser le système de mots de passe intégré de 4D. Vous pouvez définir un “Utilisateur Web générique” pour simplifier la gestion des accès à l’intérieur de la base.
La **Méthode base Sur authentification Web** vous permet d'évaluer toute requête avant qu'elle ne soit traitée par le serveur Web. La définition d’un dossier racine HTML par défaut vous permet de verrouiller les accès aux fichiers sur le disque.
Enfin, vous devez désigner individuellement les méthodes projet pouvant être exécutées via le Web.

- **Connexions SSL**

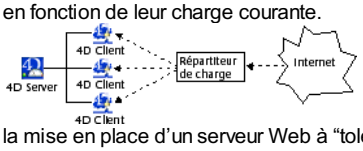
Le serveur Web 4D peut communiquer en mode sécurisé avec les navigateurs Web, à l'aide du protocole SSL (Secured Socket Layer). Ce protocole, compatible avec la majorité des navigateurs Web, permet d'authentifier les intervenants et garantit la confidentialité et l'intégrité de l'information échangée.

- **Support étendu des formats Internet**

Le serveur Web 4D est compatible HTTP/1.1, il peut gérer des documents XML et supporte la technologie WML (Wireless Markup Language).
Le serveur Web 4D prend en charge la compression GZIP de manière étendue : à l’issue d’une "négociation" entre le serveur et le client Web, tous les échanges peuvent être potentiellement compressés, afin d'obtenir les meilleures performances.

- **Exploitation simultanée des bases de données**
  - **4D en mode local et le Web**
    Lorsqu'une base 4D est publiée sur le Web avec 4D en mode local, il est possible, simultanément :
    - d'exploiter la base localement avec 4D
    - de se connecter à la base avec un navigateur Web
  - **4D Server et le Web**
    Lorsqu'une base 4D est publiée sur le Web avec 4D Server, il est possible de se connecter à la base 4D et de l'utiliser simultanément :
    - à partir de postes 4D distants
    - à partir de navigateurs Web
  - **4D en mode distant et le Web**
    Lorsqu'une base 4D est publiée sur le Web avec un client 4D, il est possible de se connecter à la base 4D et de l'utiliser simultanément :
    - à partir de postes 4D distants
    - à partir de navigateurs Web. Dans ce cas, si la base est également publiée avec 4D Server, les navigateurs Web peuvent se connecter à la base publiée via un poste 4D client ou via 4D Server. Ce fonctionnement permet notamment de gérer des modes d'accès différents aux données (public, administration, etc.).
    Les mécanismes élémentaires du serveur Web de 4D sont exploités de manière semblable par 4D en mode distant. Le fonctionnement des commandes de langage est généralement identique, que la commande soit exécutée sur 4D en mode local, 4D Server ou 4D en mode distant. Le principe est que les commandes sont appliquées au site Web du poste sur lequel elles sont exécutées. Vous devez gérer ce principe à l’aide des commandes Exécuter sur serveur / Exécuter sur Client.

- **Répartition de charge avec les clients 4D** : tout poste 4D en mode distant pouvant être utilisé comme serveur Web, vous pouvez mettre en place un système de serveur Web dynamique avec répartiteur de charge. Les possibilités offertes sont vastes, notamment :
  - la mise en place d’un système de répartition de charge (load balancing) afin d’optimiser les performances du serveur Web 4D : une réplique du site Web étant installée sur chaque serveur Web de 4D, un répartiteur de charge (matériel ou logiciel) adressera les requêtes aux postes clients en fonction de leur charge courante.



  - la mise en place d’un serveur Web à “tolérance de panne” : le site Web 4D est répliqué sur deux ou plusieurs postes clients 4D. En cas de défaillance d’un serveur Web 4D, un autre prend le relais.
  - la création de vues différentes des mêmes données, par exemple en fonction de la provenance des requêtes. Dans le cadre d’un réseau d’entreprise, un serveur Web de poste client 4D protégé peut servir les requêtes Intranet et un autre serveur Web de poste client 4D, situé au-delà du firewall, sert les requêtes Internet.

- la répartition des tâches entre les différents serveurs Web des clients 4D : un serveur Web de 4D peut être chargé des requêtes SOAP, un autre des requêtes standard, etc.

Mise en route du serveur Web et gestion des connexions

4D et 4D Server contiennent un serveur Web (aussi appelé serveur HTTP) vous permettant de publier de manière transparente et dynamique les données de vos bases sur le Web.
Cette section présente les étapes nécessaires au lancement du serveur Web et à la connexion de navigateurs, ainsi que les process de gestion des connexions.

Conditions de publication d'une base 4D sur le Web

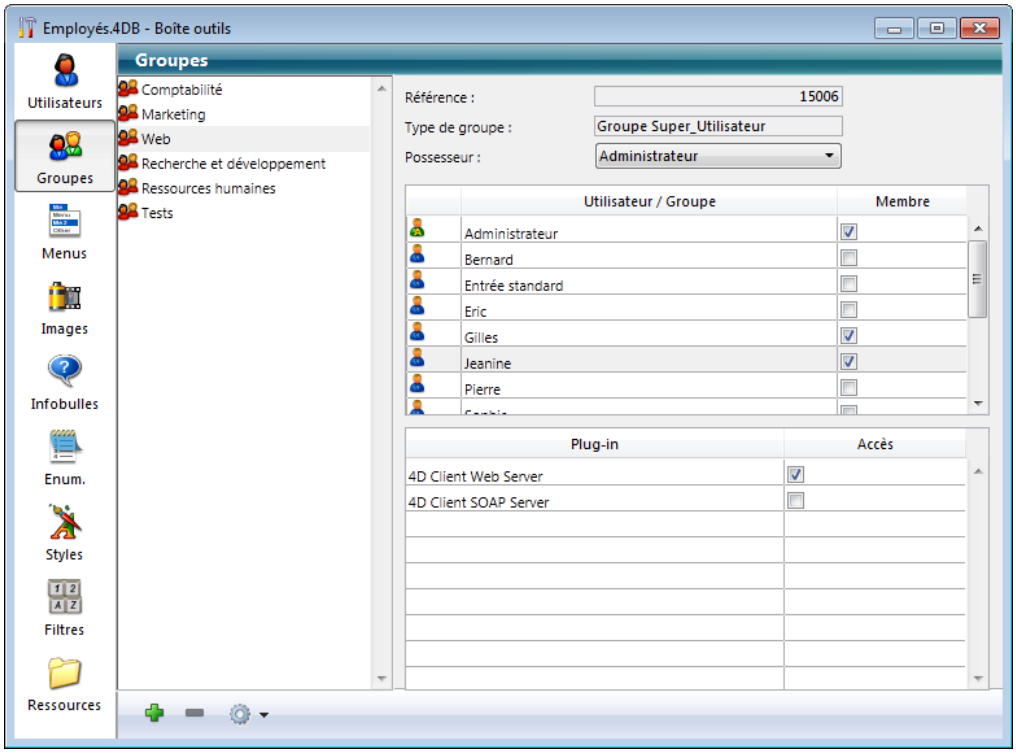
Pour pouvoir lancer le serveur HTTP de 4D ou de 4D Server, vous devez disposer des éléments décrits ci-dessous :

- une licence "4D Web Application". Pour plus d'informations sur ce point, reportez-vous au guide d'installation de 4D.
- les connexions Web sont effectuées par le biais du protocole réseau TCP/IP. Par conséquent :
  - le protocole TCP/IP doit être installé et correctement configuré sur votre machine. Reportez-vous à la documentation de votre ordinateur ou de votre système d'exploitation pour plus d'informations sur ce point.
  - Si vous voulez utiliser le protocole SSL pour vos connexions, assurez-vous que les composants nécessaires sont correctement installés (reportez-vous à la section **Utiliser le protocole TLS**).
- Une fois les points précédents contrôlés et réglés, vous devez démarrer le serveur Web depuis 4D. Ce point est traité plus loin dans cette section.

Autorisation de publication (4D en mode distant)

Par défaut, tout poste client 4D peut publier sur le Web la base à laquelle il est connecté. Vous pouvez toutefois contrôler la possibilité de publication Web de chaque poste 4D distant en utilisant le système de mots de passe de 4D.
En effet, les licences Web 4D sont considérées par 4D Server comme des licences de plug-ins. Ainsi, comme pour un plug-in, vous pouvez restreindre le droit d'utiliser les licences Web Server à un groupe d'utilisateurs spécifique.

Pour cela, affichez la page **Groupes** dans la Boîte à outils depuis 4D (vous devez disposer des autorisations d'accès adéquates pour modifier ces paramètres).
Sélectionnez un groupe dans la liste de gauche, puis cochez l'option **Accès** en regard de la ligne **4D Client Web Server** dans la zone de répartition des plug-ins :



Ci-dessus : seuls les utilisateurs appartenant au groupe “Web” sont autorisés à publier leur 4D en tant que serveur Web.

HelperTool sous Mac OS X

Sous Mac OS X, l'utilisation des ports TCP/IP réservés à la publication Web (ports 0 à 1023) requiert des privilèges d'accès spécifiques. Pour que vous puissiez utiliser ces ports, 4D fournit un programme utilitaire nommé HelperTool. Lorsque ce programme est installé, il récupère les privilèges adéquats et prend automatiquement en charge l'ouverture des ports Web. Ce mécanisme fonctionne avec 4D (tous modes), 4D Server et les applications exécutables 4D Volume Desktop.

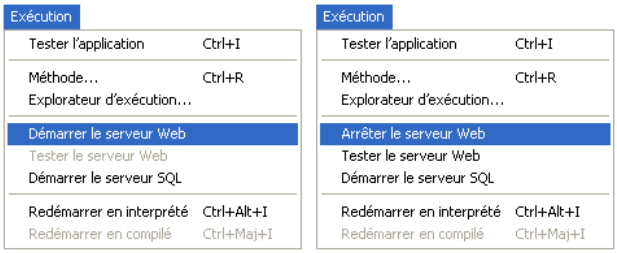
L'application HelperTool est incluse dans le progiciel 4D. L'installation s'effectue automatiquement lors de la première ouverture d'un port de numéro <1024 sur le poste. L'utilisateur est informé qu'un outil va être installé et est invité à saisir un nom et un mot de passe d'administrateur de la machine. Cette opération n'a lieu qu'une seule fois.
L'application est renommée "com.4D.HelperTool" et est installée dans le dossier "/Library/PrivilegedHelperTools/". Après la séquence initiale, le serveur Web de 4D peut être démarré et stoppé de façon transparente, quelle que soit la version de 4D.

Démarrer le serveur Web 4D

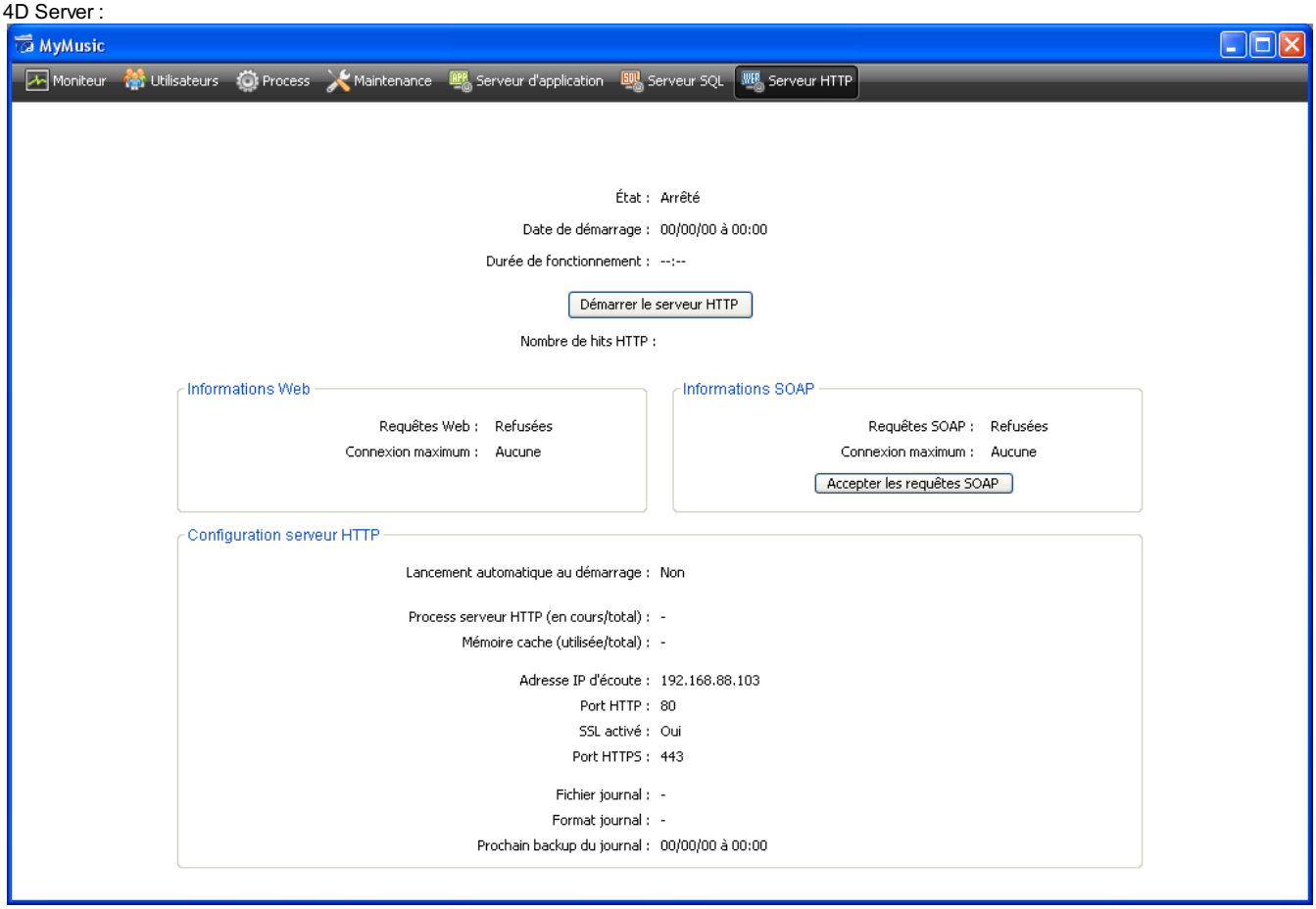
Le serveur Web 4D peut être démarré de trois manières différentes :

- par l'intermédiaire du menu **Exécution** de 4D ou de la page Serveur HTTP de 4D Server (bouton **Démarrer le serveur HTTP**). Ces commandes vous permettent de lancer et d'arrêter le serveur Web à tout moment :

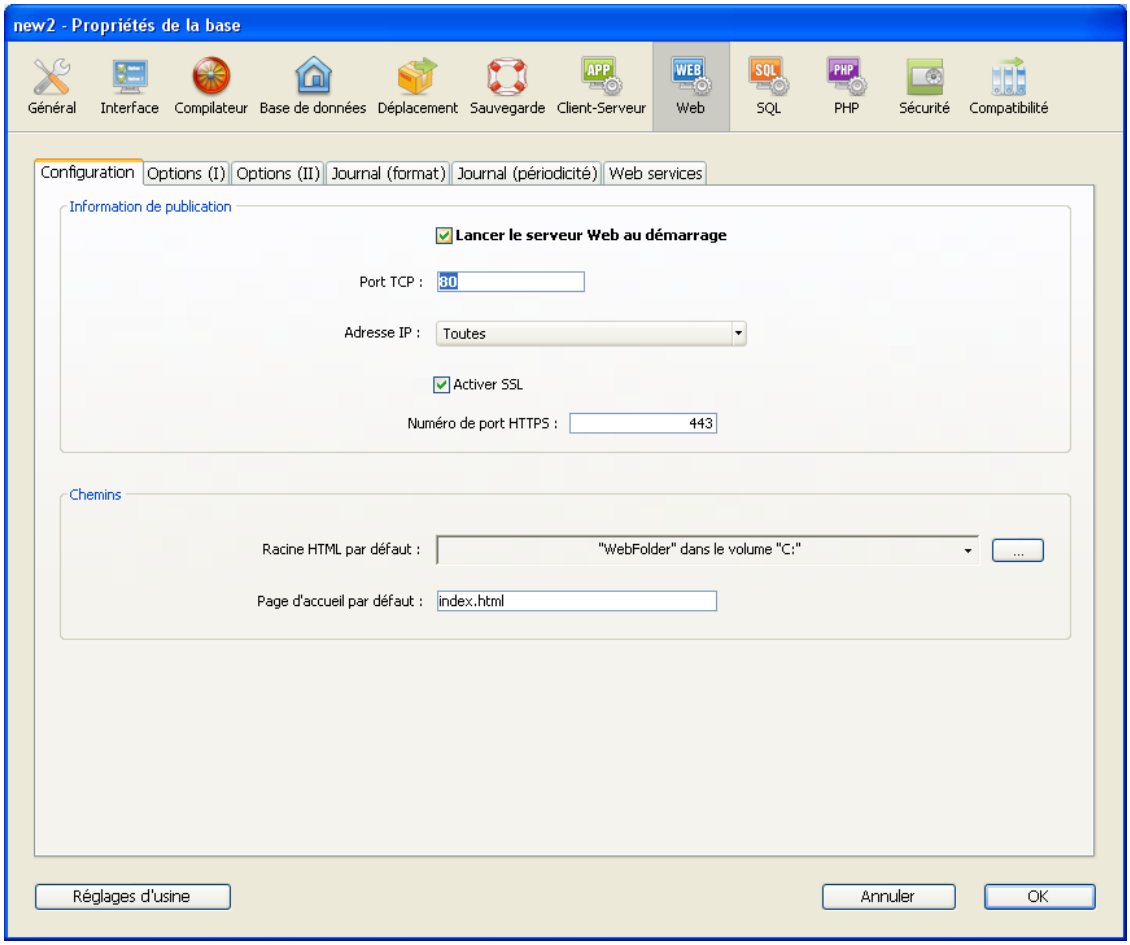
4D :



4D Server :



- par le démarrage automatique à chaque ouverture de l'application 4D. Pour cela, affichez la page **Configuration** du thème **Web** des Propriétés de la base :



Dans la zone “Information de publication”, cochez la case **Lancer le serveur Web au démarrage** puis cliquez sur le bouton **OK**. La base sera désormais automatiquement publiée comme serveur Web chaque fois que vous l'ouvrirez avec 4D ou 4D Server.

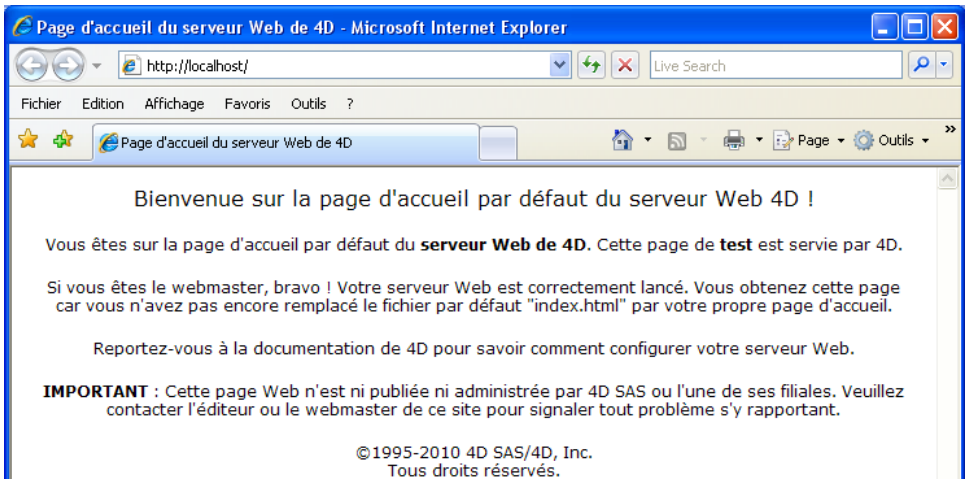
- par programmation, en appelant la commande **WEB DEMARRER SERVEUR**.

Note : Il n'est pas nécessaire de rouvrir votre base de données pour lancer ou arrêter sa publication comme serveur Web. Vous pouvez interrompre et redémarrer le serveur Web autant de fois que vous voulez à l'aide du menu **Exécution**, du bouton **Démarrer le serveur HTTP ou en appelant les commandes WEB DEMARRER SERVEUR** et **WEB ARRETER SERVEUR**.

Tester le serveur Web

La commande **Tester le serveur Web** permet de contrôler le fonctionnement du serveur Web intégré (4D uniquement). Cette commande est accessible dans le menu **Exécution** lorsque le serveur Web est lancé :

Lorsque vous sélectionnez cette commande, la page d’accueil du site Web publié par l’application 4D s’affiche dans une fenêtre de votre navigateur par défaut :



Cette commande permet de vérifier le fonctionnement du serveur Web, l'affichage de la page d'accueil, etc. La page est appelée via l'URL Localhost, qui est le raccourci standard désignant l'adresse IP de la machine sur laquelle est exécuté le navigateur. La commande tient compte du numéro de port TCP de publication spécifié dans les Propriétés de l'application.

Connexion à une base 4D publiée sur le Web

Une fois que vous avez lancé la publication d'une base 4D sur le Web, vous pouvez vous y connecter avec un navigateur Web. Pour cela :

- Si votre site Web dispose d'un nom d'hôte enregistré (par exemple, "*www.bellesfleurs.com*"), il vous suffit d'indiquer ce nom dans la zone "Ouvrir", "Adresse" ou encore "Lieu" du navigateur puis d'appuyer sur la touche **Entrée** pour vous connecter.

- Si votre site Web ne dispose pas d'un nom enregistré, indiquez l'adresse IP de la machine de la base (par exemple 123.4.567.89) dans la zone "Ouvrir", "Adresse" ou encore "Lieu" du navigateur puis appuyez sur la touche **Entrée**.

A cet instant, votre navigateur doit afficher la page d'accueil de votre site Web. Si vous avez publié une base en conservant les paramétrages standard, vous devez obtenir la page d'accueil par défaut du serveur Web de 4D. Cette page vous permet de tester la connexion et le fonctionnement du serveur.

Vous pouvez également rencontrer une des situations décrites ci-dessous.

Note : Si votre base est protégée par un système de contrôle d'accès, il se peut que vous ayez à saisir un nom et un mot de passe (pour plus d'informations, reportez-vous à la section **Sécurité des connexions**).

(1) La connexion échoue, vous obtenez un message du type "...le serveur n'accepte pas de connexions ou est occupé...". Dans ce cas, effectuez les contrôles suivants :

- Vérifiez que le nom du serveur ou l'adresse IP que vous avez saisi(e) est correct(e).
- Vérifiez que 4D ou 4D Server est bien lancé et que le serveur Web a bien démarré.
- Vérifiez que la base de données est bien configurée pour être publiée sur le port TCP Web par défaut, c'est-à-dire 80 (voir aussi point 3).
- Vérifiez que le protocole réseau TCP/IP est correctement configuré sur la machine serveur et sur la machine du navigateur (les deux machines doivent se trouver sur le même réseau ou sous-réseau, ou les routeurs doivent être correctement configurés).
- Vérifiez les connexions physiques.
- Si vous ne testez pas localement votre propre site mais essayez de vous connecter à une base Web publiée sur Internet ou Intranet par quelqu'un d'autre, il se peut qu'en définitive le message décrive une situation réelle : le poste serveur peut être éteint ou occupé, dans ce cas vous pouvez tenter de vous connecter ultérieurement ou contacter l'administrateur du site Web.

(2) La connexion est établie, mais vous obtenez une erreur HTTP 404, "Fichier non trouvé". Ce cas signifie que la page d'accueil du site n'a pu être servie. Dans ce cas, vérifiez que la page d'accueil existe bien à l'emplacement défini dans les Propriétés de la base (cf. section **Paramétrages du serveur Web**) ou à l'aide de la commande **WEB FIXER PAGE ACCUEIL**.
(3) La connexion est établie, mais vous n'obtenez pas la page Web que vous attendiez ! Cela peut se produire lorsque plusieurs serveurs Web sont exécutés simultanément sur la même machine. Par exemple :

- Vous avez lancé une seule base Web 4D, mais sur un système Windows qui exécute déjà son propre serveur Web.
- Vous avez lancé plusieurs bases Web 4D sur la même machine.

Dans les cas décrits ci-dessus, il vous suffit de changer les numéros des ports TCP sur lesquels vos bases 4D Web sont publiées. Pour cela, reportez-vous à la section **Paramétrages du serveur Web**.

Process Web

Chaque fois qu'un navigateur Web tente de se connecter à la base, la requête est gérée de la manière suivante :

- D'abord, un ou plusieurs process 4D locaux appelés **Process de connexion Web** sont créés pour gérer et évaluer la connexion au navigateur Web. Ces process gèrent toutes les requêtes HTTP. Ils s'exécutent très rapidement puis sont "tués" ou endormis. En effet, à des fins d'optimisation du serveur Web, une fois qu'il a traité une requête, un process de connexion Web temporaire est gelé pendant quelques secondes pour être éventuellement réactivé lorsqu'une autre requête arrive. Ce mécanisme peut être ajusté (délai d'attente, nombre minimum et maximum de process à conserver dans la "réserve" de process) à l'aide de la commande **FIXER PARAMETRE BASE**.

- Le process Web prend alors en charge le traitement de la requête et l'envoi de la réponse éventuelle au navigateur. Le process temporaire est alors tué ou endormi (cf. ci-dessus).

Prise en charge d'IP v6

A compter de la version 14, 4D prend en charge la notation d'adresses IPv6. Les serveurs suivants de 4D sont concernés :

- le serveur Web ainsi que le serveur SOAP,
- le serveur SQL.

Note : Pour plus d'informations sur IPv6, reportez-vous à la spécification RFC 2460.

La prise en charge d'IPv6 est transparente pour les utilisateurs et pour les développeurs 4D : le programme accepte indifféremment les connexions IPv6 ou IPv4 lorsque la configuration "Adresse IP" d'écoute du serveur est sur **Toutes** (cf. **Adresse IP** (serveur HTTP) et **Préférences de publication du serveur SQL** (serveur SQL).

Il convient toutefois de prêter attention aux points suivants :

- **Indication des numéros de port**
  La notation IP v6 utilisant des caractères deux-points (:), l'ajout de numéros de port peut introduire une certaine confusion, par exemple :

  ```
  2001:0DB8::85a3:0:ac1f:8001 //adresse IPv6
  2001:0DB8::85a3:0:ac1f:8001:8081 //adresse IPv6 port 8081
  ```

  Afin de limiter cette confusion, il est recommandé d'utiliser la notation [ ] lorsqu'une adresse IPv6 est combinée à un numéro de port, par exemple :

  ```
  [2001:0DB8::85a3:0:ac1f:8001]:8081 //adresse IPv6 port 8081
  ```

- **Non détection de l'occupation du port TCP**
  Lorsque le serveur répond sur "toutes" les adresses IP, l'occupation du port TCP par une autre application n'est pas signalée au démarrage du serveur (à la différence des versions précédentes de 4D). En effet dans ce cas, le serveur 4D ne détecte pas d'erreur car le port reste libre sur l'adresse IPv6. Cependant, il n'est pas possible d'y accéder via l'adresse IPv4 de la machine ni l'adresse locale 127.0.0.1.
  Si votre serveur 4D semble ne pas répondre sur le port défini, vous pouvez tester l'adresse [::1] sur la machine du serveur (équivaut sous IPv6 à 127.0.0.1, ajoutez :*numPort* pour tester un numéro de port autre que celui par défaut). Si 4D répond, il est probable qu'une autre application occupe le port en IPv4.

- **Adresses IPv6 basées sur IPv4**
  4D propose une représentation hybride standard des adresses IPv4 en IPv6, permettant de standardiser les traitements. Ces adresses ont un préfixe de 96 bits au format IPv6, suivi de 32 bits écrits dans la notation décimale à points de IPv4. Par exemple, ::ffff:192.168.2.34 représente l'adresse IPv4 192.168.2.34.

Sécurité des connexions

Vous pouvez sécuriser les connexions à votre serveur Web 4D à l'aide des éléments suivants :

- Pour les accès Web, la combinaison du système de gestion des mots de passe (mode BASIC ou mode DIGEST) et de la **Méthode base Sur authentification Web**,
- La définition d'un "Utilisateur Web générique",
- La définition d'un dossier racine HTML par défaut,
- La définition de la propriété "Disponible via les balises et les URLs 4D (4DACTION…)" pour chaque méthode projet de la base,
- L'option de prise en charge spécifique des requêtes de synchronisation via HTTP.

Note : La sécurité des informations transmises via la connexion est prise en charge par le protocole SSL. Pour plus d'informations, reportez-vous à la section **Utiliser le protocole TLS**

Gestion des mots de passe pour les accès Web

Mode BASIC et Mode DIGEST

Vous pouvez définir, dans les Propriétés de la base, le système de contrôle d'accès que vous souhaitez appliquer à votre serveur Web. Deux modes d'authentification sont proposés : le mode BASIC et le mode DIGEST. Le mode d'authentification concerne la manière dont sont collectées et traitées les informations relatives au nom d'utilisateur et au mot de passe.

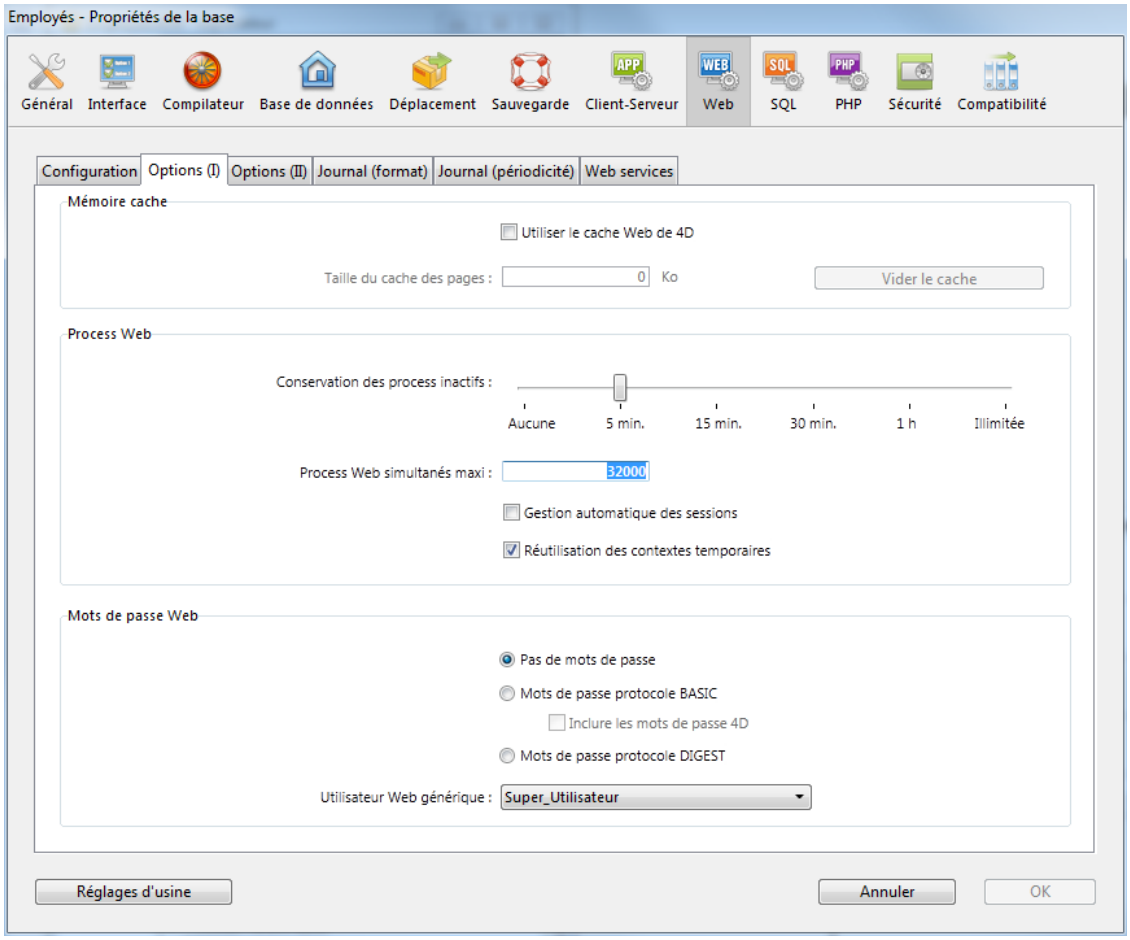
- En mode BASIC, le nom et le mot de passe saisis par l'utilisateur sont envoyés en clair dans les requêtes HTTP. Ce principe n'assure pas une sécurité totale au système dans la mesure où ces informations peuvent être interceptées et utilisées par un tiers.
- Le mode DIGEST procure un niveau de sécurité plus élevé car les informations d'authentification sont traitées par un processus unidirectionnel appelé hachage qui rend leur contenu impossible à décrypter.

Côté utilisateur, l'emploi d'un mode d'authentification ou de l'autre est transparente.

Notes :

- Pour des raisons de compatibilité, le mode d'authentification BASIC est utilisé par défaut dans les bases 4D converties en version 11 (si l'option "Utiliser mots de passe" était cochée dans la version précédente). Vous devez activer explicitement le mode Digest.
- L'authentification Digest est une fonction de HTTP1.1 et n'est pas prise en charge par tous les navigateurs. Par exemple, seules les versions 5.0 et suivantes du navigateur Microsoft Internet Explorer acceptent ce mode. Si un navigateur ne prenant pas en charge cette fonctionnalité adresse une demande au serveur Web alors que l'authentification Digest est activée, le serveur rejette la demande et retourne un message d'erreur au navigateur.

Le choix du mode d'authentification s'effectue dans la page Web/Options (I) de la boîte de dialogue des Propriétés de la base :



Dans la zone "Mots de passe", vous disposez de trois options :

- **Pas de mots de passe** : aucune authentification n'est effectuée pour la connexion au serveur Web. Dans ce cas :

    - Si la **Méthode base Sur authentification Web** existe, elle est exécutée et, outre $1 et $2, seules les adresses IP du navigateur et du serveur ($3 et $4) sont renseignées, le nom d'utilisateur et le mot de passe ($5 et $6) sont vides. Vous pouvez dans ce cas filtrer les connexions en fonction de l'adresse IP du navigateur et/ou de l'adresse IP demandée du serveur.
    - Si la **Méthode base Sur authentification Web** n'existe pas, les connexions sont automatiquement acceptées.
- **Mots de passe protocole BASIC** : authentification standard en mode BASIC. Lorsqu'un utilisateur se connecte, une boîte de dialogue lui permettant de saisir son nom et son mot de passe s'affiche sur son navigateur. Ces deux valeurs sont ensuite envoyées en clair à la **Méthode base Sur authentification Web** avec les autres paramètres de la connexion (adresse et port IP, URL...) pour que vous puissiez les traiter.
  Ce mode donne accès à l'option **Inclure les mots de passe 4D**, permettant d'utiliser, au lieu ou en plus de votre propre système, le système 4D de mots de passe de la base (défini dans 4D).
- **Mots de passe protocole DIGEST** : authentification en mode DIGEST. Comme en mode BASIC, l'utilisateur doit saisir son nom et son mot de passe lorsqu'il se connecte. Ces deux valeurs sont alors envoyées cryptées à la **Méthode base Sur authentification Web** avec les autres paramètres de la connexion. Vous devez authentifier l'utilisateur à l'aide de la commande **WEB Valider digest**.

Notes :

- Vous devez redémarrer le serveur Web pour que les modifications effectuées sur ces paramètres soient prises en compte
- Avec le serveur Web de 4D Client, il est à noter que tous les sites publiés par les postes 4D Client partageront la même table d'utilisateurs. En effet, la validation des utilisateurs/mots de passe est effectuée par l'application 4D Server.

Mode BASIC : Combinaison des mots de passe et de la Méthode base Sur authentification Web

Si vous utilisez le mode BASIC, le système de filtrage des connexions au serveur Web de 4D dépend de la combinaison de deux paramètres :

- les options de mots de passe Web dans la boîte de dialogue des Propriétés de la base,
- l'existence ou non de la **Méthode base Sur authentification Web**.

Voici les différentes possibilités de contrôle des connexions :

L'option "Mots de passe protocole BASIC" est cochée et l'option "Inclure les mots de passe de 4D" n'est pas cochée

- Si la **Méthode base Sur authentification Web** existe, elle est exécutée et tous ses paramètres sont renseignés. Vous pouvez dans ce cas filtrer très précisément les connexions à partir du nom d'utilisateur, du mot de passe et/ou des adresses IP du navigateur et du serveur Web.
- Si la **Méthode base Sur authentification Web** n'existe pas, la connexion est automatiquement refusée et un message indiquant que la méthode d'authentification n'existe pas est envoyé au navigateur.

Note : Si le nom d'utilisateur envoyé est une chaîne vide et si la **Méthode base Sur authentification Web** n'existe pas, une boîte de dialogue de demande de mot de passe est envoyée au navigateur.

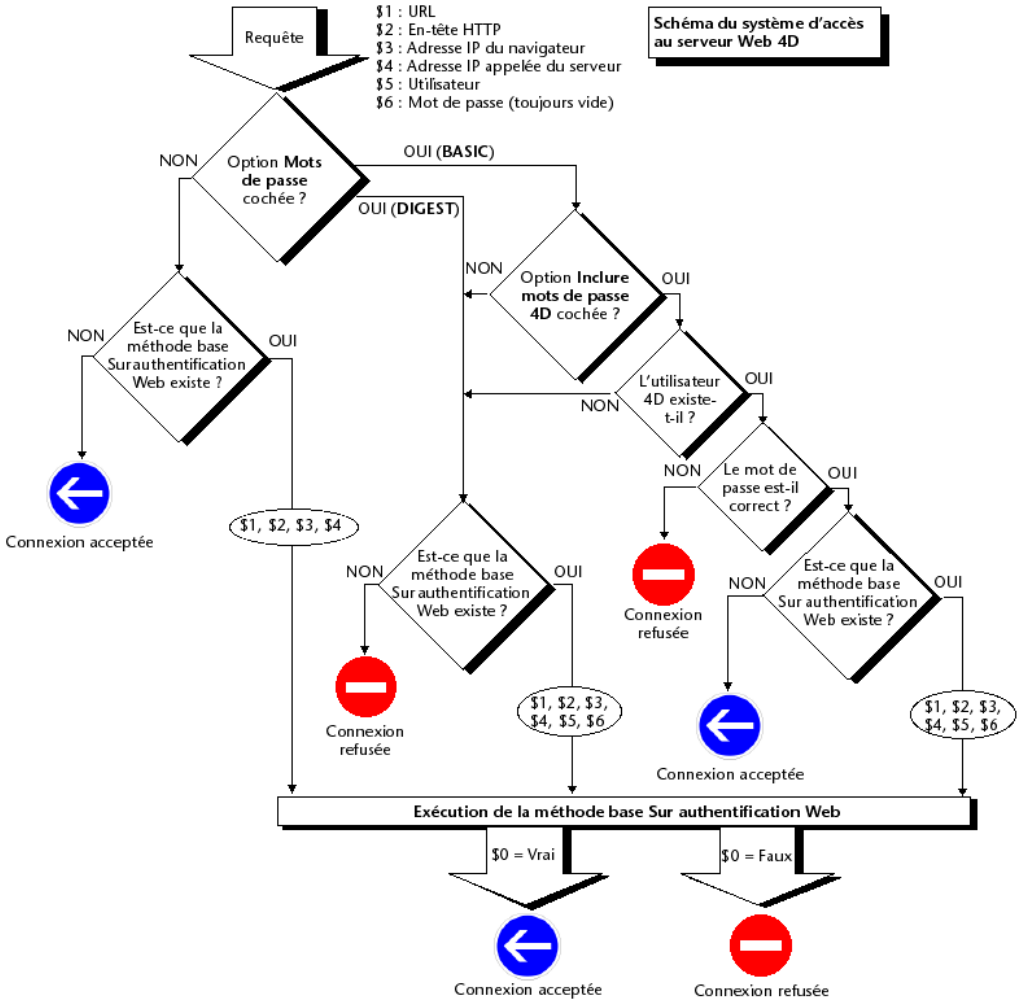
Les options "Mots de passe protocole BASIC" et "Inclure les mots de passe de 4D" sont cochées

- Si le nom d'utilisateur envoyé par le navigateur existe dans la table des utilisateurs 4D et que le mot de passe est valide, la connexion est acceptée (dans ce cas, pour des raisons de sécurité le paramètre $6 n'est alors pas renseigné). Si le mot de passe est invalide, la connexion est refusée.
- Si le nom d'utilisateur envoyé par le navigateur n'existe pas dans 4D, deux cas sont alors possibles :
    - si la **Méthode base Sur authentification Web** existe, les paramètres $1, $2, $3, $4, $5 et $6 sont renseignés. Vous pouvez dans ce cas filtrer très précisément les connexions à partir du nom d'utilisateur, du mot de passe et/ou des adresses IP du navigateur et du serveur Web.
    - si la **Méthode base Sur authentification Web** n'existe pas, la connexion est refusée.

Mode DIGEST

A la différence du mode BASIC, le mode DIGEST n'est pas compatible avec les mots de passe 4D standard : il n'est pas possible d'utiliser les mots de passe 4D comme identifiants Web. L'option "Inclure les mots de passe 4D" est grisée lorsque ce mode est sélectionné. Les identifiants des utilisateurs Web doivent être gérés de façon personnalisée (par exemple via une table).
Lorsque le mode DIGEST est activé, le paramètre $6 (mot de passe) est toujours retourné vide dans la **Méthode base Sur authentification Web**. En effet dans ce mode, cette information ne transite pas en clair par le réseau. Vous devez impérativement dans ce cas évaluer la demande de connexion à l'aide de la commande **WEB Valider digest**.

Le fonctionnement du système d'accès au serveur Web 4D est résumé dans le schéma suivant :



A propos des robots (note de sécurité)

Certains robots (moteurs de recherche, spiders) parcourent les serveurs Web et les pages statiques. Si vous souhaitez que les robots ne puissent pas accéder à la totalité de votre site, il est possible de définir des URL qui leur seront interdits.
Pour cela, placez un fichier nommé ROBOTS.TXT à la racine du serveur. Ce fichier doit être structuré de la manière suivante :

```
User-Agent: <nom>
Disallow: <URL> ou <début d'URL>
```

Par exemple :

```
User-Agent: *
Disallow: /4D
Disallow: /%23%23
Disallow: /GIFS/
```

"User-Agent: *" signifie qu'il s'agit de tous les robots.
"Disallow: /4D" signifie que les robots ne doivent pas accéder aux URL commençant par /4D.
"Disallow: /%23%23" signifie que les robots ne doivent pas accéder aux URL commençant par /%23%23.
"Disallow: /GIFS/' signifie que les robots ne doivent pas accéder au dossier /GIFS/ ni aux sous-dossiers.

Autre exemple :

```
User-Agent: *
Disallow: /
```

Dans ce cas, la totalité du site est interdite aux robots.

Utilisateur Web générique

Vous pouvez désigner un utilisateur — préalablement défini dans la table des mots de passe de 4D — comme "Utilisateur Web générique". Dans ce cas, chaque navigateur se connectant à la base bénéficie des autorisations et restrictions d'accès associées à cet utilisateur. Vous pouvez ainsi contrôler simplement l'accès des navigateurs aux différentes parties de la base.

Note : Il ne faut pas confondre cette option, permettant de restreindre les accès des navigateurs aux différentes parties de l'application (méthodes, formulaires, etc.), avec le système de contrôle des connexions au serveur Web, géré par les mots de passe et la **Méthode base Sur authentification Web**.

Pour définir un Utilisateur Web générique :

1. En mode Développement, créez au moins un utilisateur dans l'Editeur de mots de passe.
   Vous pouvez lui associer ou non un mot de passe.

2. Dans les différents éditeurs de 4D, assignez à cet utilisateur les autorisations et restrictions d'accès souhaitées.
3. Dans la boîte de dialogue des Propriétés, choisissez le thème **Web**, page **Options (I)**.
   La zone "Mots de passe Web" contient la liste déroulante **Utilisateur Web générique**. Par défaut, l'utilisateur Web générique est le Super_Utilisateur : les navigateurs disposent donc d'un accès libre à toutes les parties de la base.
4. Choisissez l'utilisateur dans la liste déroulante et validez la boîte de dialogue.

Tous les navigateurs Web autorisés à se connecter à la base bénéficieront des autorisations et restrictions d'accès associées à l'utilisateur Web générique (sauf lorsque le mode BASIC et l'option "Inclure les mots de passe 4D" sont cochés et que l'utilisateur qui se connecte existe dans la table des mots de passe 4D, cf. ci-dessous).

Interaction avec le protocole BASIC

L'option "Mots de passe protocole BASIC" n'influe pas sur le mécanisme de l'utilisateur Web générique : quel que soit l'état de cette option, les privilèges et restrictions d'accès associés à l'"Utilisateur Web générique" seront appliqués à tous les navigateurs Web autorisés à se connecter à la base.

En revanche, lorsque l'option "Inclure les mots de passe 4D" est cochée, deux cas peuvent se produire :

- Le nom et le mot de passe de l'utilisateur n'existent pas dans la table des mots de passe de 4D. Dans ce cas, si la connexion est acceptée par la **Méthode base Sur authentification Web**, les droits d'accès de l'utilisateur Web générique seront appliqués au navigateur.
- Le nom et le mot de passe de l'utilisateur existent dans la table des mots de passe de 4D. Dans ce cas, le paramètre "Utilisateur Web générique" est ignoré : l'utilisateur se connecte avec ses propres droits d'accès.

Racine HTML par défaut

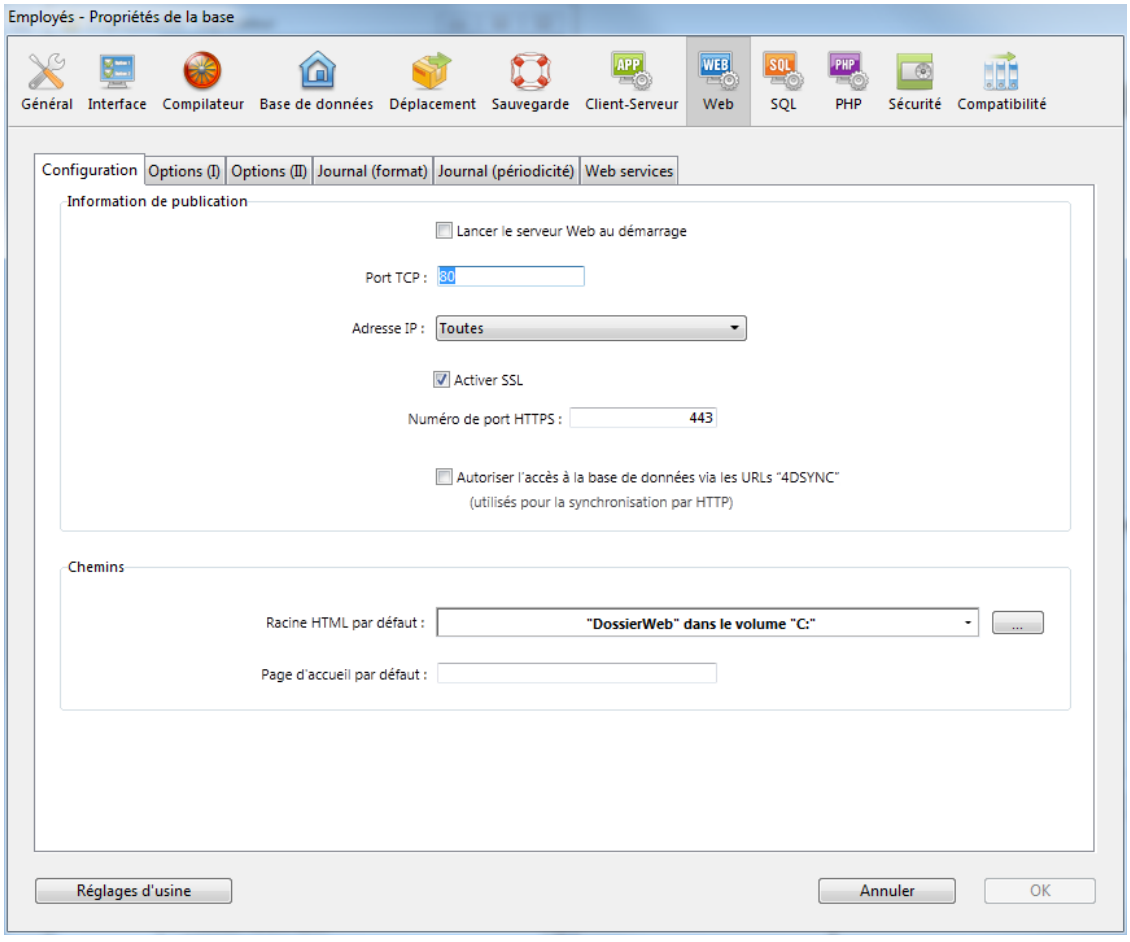
Cette option des Propriétés de la base vous permet de définir le dossier dans lequel 4D recherchera les pages HTML statiques et semi-dynamiques, les images, etc., à envoyer aux navigateurs.

De plus, le dossier racine HTML définit, sur le disque dur du serveur Web, le niveau hiérarchique au-dessus duquel les fichiers ne seront pas accessibles. Cette restriction d'accès s'applique aux URLs demandés par les navigateurs Web ainsi qu'aux commandes du serveur Web 4D telles que **WEB ENVOYER FICHIER**. Si un URL demandé par un navigateur ou une commande 4D tente d'accéder à un fichier situé en amont du dossier racine HTML, une erreur est retournée, indiquant que le fichier n'a pas été trouvé.

Par défaut, 4D définit un dossier racine HTML intitulé **DossierWeb**. S'il n'existe pas déjà, le dossier racine HTML est créé sur le disque au premier lancement du serveur Web.
Si vous conservez l'emplacement par défaut, le dossier racine est créé :

- avec 4D en mode local et 4D Server, au même niveau que le fichier de structure de la base.
  Note : dans le cadre d'une application compilée et fusionnée, le fichier de structure est placé dans le sous-dossier **Database**.
- avec 4D en mode distant, dans le dossier local de la base 4D (cf. commande **Dossier 4D**).

Vous pouvez modifier le nom et l'emplacement du dossier racine HTML par défaut dans la boîte de dialogue des Propriétés de la base (thème **Web**, page **Configuration**) :



Dans la zone "Racine HTML par défaut", saisissez le nouveau chemin d'accès du dossier que vous souhaitez utiliser.
Le chemin d'accès saisi dans cette boîte de dialogue est relatif : il est établi à partir du dossier contenant la structure de la base (4D en mode local ou 4D Server) ou du dossier contenant l'application ou le progiciel 4D (4D en mode distant).
Afin d'assurer la compatibilité multiplate-forme de vos bases, le serveur Web 4D utilise, pour décrire les chemins d'accès, des conventions d'écriture particulières. Les règles de syntaxe sont les suivantes :

- les dossiers sont séparés par le caractère /

- le chemin ne doit pas se terminer par /
- pour “remonter” d’un niveau dans la hiérarchie des dossiers, saisissez “..” (point point) devant le nom du dossier,
- le chemin ne doit pas commencer par / (sauf si vous souhaitez que le dossier racine HTML soit le dossier de la base ou du 4D distant, cf. ci-dessous).

Par exemple, si vous souhaitez que le dossier racine HTML soit le sous-dossier “Web”, placé dans le dossier “Base4D”, saisissez **Bases4D/Web**
Si vous souhaitez que le dossier racine HTML soit le dossier de la base ou du 4D distant, mais que l’accès aux dossiers des niveaux supérieurs soit interdit, saisissez / dans la zone. Pour que l’accès aux volumes soit totalement libre, laissez la zone “Racine HTML par défaut” vide.

ATTENTION : Si vous ne définissez aucun dossier racine HTML par défaut, le dossier contenant le fichier de structure de la base ou l'application 4D est utilisé. **Dans ce cas, il n’y a pas de restrictions d’accès** (tous les volumes sont accessibles).

Notes :

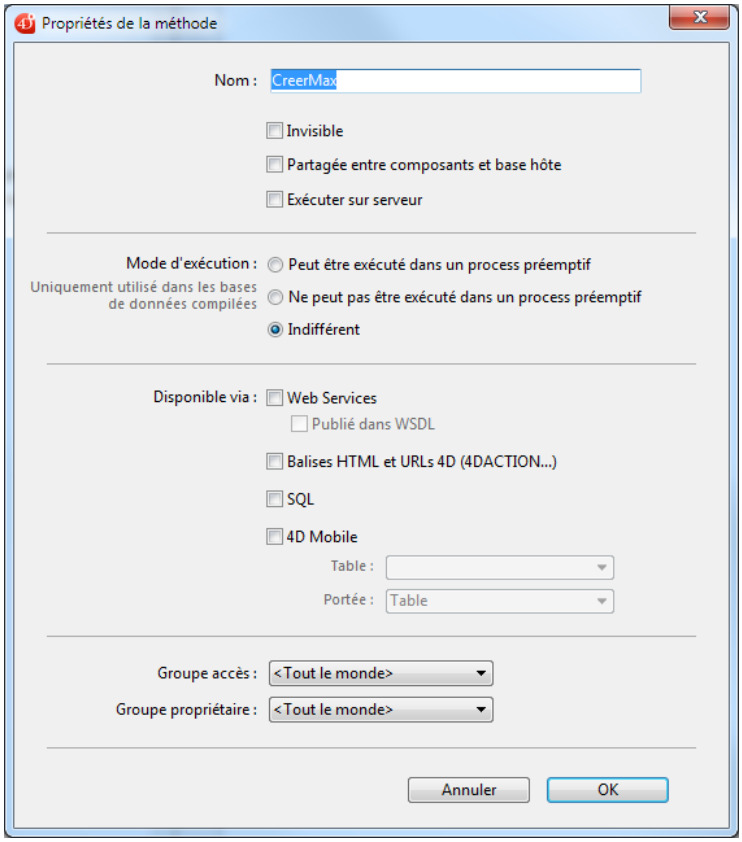
- Lorsque le dossier racine HTML est modifié dans les Propriétés de la base, le cache est effacé afin de ne pas conserver des fichiers dont l’accès serait devenu restreint.
- Il est possible de définir dynamiquement le dossier racine HTML par défaut à l’aide de la commande **WEB FIXER RACINE**. Dans ce cas, la modification s’applique à tous les process Web courants pour la session de travail. Le cache des pages HTML est alors effacé.

Disponible pour les balises et les URLS 4D

L'URL spécial *4DACTION*, les balises *4DSCRIPT*, *4DEVAL*, *4DTEXT, 4DHTML* (ainsi que les anciennes balises *4DVAR* et *4DHTMLVAR*) permettent de déclencher l'exécution de toute méthode projet d’une base 4D publiée sur le Web. Par exemple, la requête *http://www.serveur.com/4DACTION/Effacer_Tout* provoque l'exécution de la méthode projet ***Effacer_Tout***, si elle existe.

Ce mécanisme présente donc un risque pour la sécurité de la base, notamment si un internaute déclenche intentionnellement ou par erreur une méthode non destinée à une exécution via le Web. Vous pouvez prévenir ce risque de trois manières :

- restreindre les accès aux méthodes projet via le système de mots de passe 4D. Inconvénients : ce système nécessite l'utilisation des mots de passe 4D et interdit tout type d'exécution de la méthode (y compris via des balises HTML).
- filtrer les méthodes appelées via des URLs à l'aide de la **Méthode base Sur authentification Web**. Inconvénients : si la base comporte de nombreuses méthodes, ce système peut être difficile à gérer.
- utiliser l'option **Disponible via les balises et les URLs 4D (4DACTION…)** affichée dans la boîte de dialogue des Propriétés des méthodes projet :



Cette option permet de désigner individuellement chaque méthode projet pouvant être appelée via l'URL spécial *4DACTION* et les balises *4DSCRIPT*, *4DEVAL*, *4DTEXT* et *4DHTML* (ainsi que *4DVAR* et *4DHTMLVAR*). Lorsqu’elle est désélectionnée, la méthode projet concernée ne peut pas être exécutée via une requête HTTP contenant un URL ou une balise spécial(e). En revanche, elle peut être exécutée à l'aide d’autres types d'appels (formules, autres méthodes, etc.).

Cette option est désélectionnée par défaut à la création d'une base. Vous devez désigner expressément les méthodes pouvant être exécutées via l'URL *4DACTION* et les balises 4D.

Dans l'Explorateur, les méthodes projet ayant cette propriété bénéficient d’une icône spécifique :

Autoriser l’accès à la base de données via les URLs 4DSYNC

Cette option de la page "Web/Configuration" des Propriétés de la base permet de contrôler la prise en charge des requêtes comportant des URLs /4DSYNC. Ces URLs sont utilisés pour la synchronisation des données via HTTP (pour plus d’informations sur ce mécanisme, reportez-vous au paragraphe **URL 4DSYNC/**).

Cette option a pour effet d’activer ou d’inactiver le traitement spécifique des requêtes contenant /4DSYNC :

- lorsqu'elle n'est pas cochée, les requêtes /4DSYNC sont considérées comme des requêtes standard et ne permettent pas de traitement spéficique (l'utilisation d'une requête de synchronisation provoque l'envoi de la réponse type 404 - ressource indisponible).
- lorsqu'elle est cochée, le mécanisme de synchronisation est activé ; les requêtes /4DSYNC sont considérées comme des requêtes spéciales et sont analysées par le serveur HTTP de 4D.

Par défaut :

- cette option est **désélectionnée** dans les bases de données créées avec 4D à partir de la version 13.
- cette option est **sélectionnée** dans les bases de données converties depuis une version de 4D antérieure à la version 13, pour des raisons de compatibilité. Il est recommandé de la désélectionner si votre application n'exploite pas la fonction de réplication par HTTP.

Cette option a une portée locale à l'application et sa prise en compte nécessite un redémarrage du serveur Web.

Méthode base Sur authentification Web

Description

La **Méthode base Sur authentification Web** est chargée de gérer les accès au moteur de serveur Web. Elle est automatiquement appelée par 4D ou 4D Server lorsqu'une requête d'un navigateur Web requiert l'exécution d'une méthode 4D sur le serveur (appel d'une méthode via un URL *4DACTION*, une balise *4DSCRIPT*, etc.).

La **Méthode base Sur authentification Web** reçoit six paramètres de type Texte, passés par 4D ($1, $2, $3, $4, $5 et $6), et retourne un booléen, $0. Voici la description de ces paramètres :

Paramètres	Type	Description
$1	Texte	URL
$2	Texte	En-tête + Corps HTTP (32 ko maximum)
$3	Texte	Adresse IP du navigateur
$4	Texte	Adresse IP appelée du serveur
$5	Texte	Nom d'utilisateur
$6	Texte	Mot de passe
$0	Booléen	Vrai = requête acceptée, Faux = requête rejetée

Vous devez déclarer ces paramètres de la manière suivante :

```
   ` Méthode base Sur authentification Web

C_TEXTE($1;$2;$3;$4;$5;$6)
C_BOOLEEN($0)

   ` Code pour la méthode
```

Note : Tous les paramètres de la **Méthode base Sur authentification Web** ne sont pas forcément remplis. Les informations reçues par la méthode base dépendent des options que vous avez sélectionnées dans la boîte de dialogue des Propriétés de la base. Référez-vous à la section **Sécurité des connexions**.

- **URL**
  Le premier paramètre (*$1*) est l'**URL** saisi par l'utilisateur dans la zone 'Adresse' de son navigateur Web, moins l'adresse hôte.
  Prenons l'exemple d'une connexion Intranet. Supposons que l'adresse **IP** de votre machine serveur Web 4D est *123.4.567.89*. Le tableau suivant liste les valeurs de *$1* selon l'**URL** saisi dans le navigateur Web :

URL saisi dans le navigateur Web	Valeur du paramètre $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients/Ajouter	/Clients/Ajouter
123.4.567.89/Faire_ceci/Si_OK/Faire_cela	/Faire_ceci/Si_OK/Faire_cela

- **En-tête et corps de la requête HTTP**
  Le deuxième paramètre (*$2*) est l'en-tête et le corps de la requête **HTTP** envoyée par le navigateur Web. Notez que ces informations sont passées telles quelles à la **Méthode base Sur authentification Web**. Le contenu varie en fonction du type de navigateur Web qui tente de se connecter. Si votre application exploite ces informations, il est de votre ressort d'analyser l'en-tête et le corps.

Notes :

- Pour des raisons de performances, la taille des données transitant via le paramètre $2 ne peut dépasser 32 Ko. Au-delà, elles sont tronquées par le serveur HTTP de 4D.
- Pour plus d'informations sur ce paramètre, reportez-vous à la description de la **Méthode base Sur connexion Web**.

- **Adresse IP du navigateur**
  Le troisième paramètre (*$3*) reçoit l'adresse IP de la machine du navigateur. Cette information peut vous permettre, en particulier, de distinguer les connexions Intranet des connexions Internet.
  Note : 4D retourne les adresses IPv4 dans un format hybride IPv6 comprenant un préfixe de 96 bits, par exemple ::ffff:192.168.2.34 pour l'adresse IPv4 192.168.2.34. Pour plus d'informations, reportez-vous à la section **Prise en charge d'IP v6**.

- **Adresse IP demandée du serveur**
  Le quatrième paramètre (*$4*) reçoit l'adresse IP demandée du serveur Web 4D. En effet, 4D autorise le multi-homing, permettant d'exploiter des machines disposant de plusieurs adresses IP. Pour plus d'informations sur ce point, reportez-vous à la section **Paramétrages du serveur Web**.

- **Nom d'utilisateur et Mot de passe**
  Les paramètres *$5* et *$6* reçoivent le nom d'utilisateur et le mot de passe saisis par l'utilisateur dans la boîte de dialogue standard d'identification affichée par le navigateur, le cas échéant.
  Cette boîte de dialogue apparaît pour chaque connexion dès qu'une option de gestion des mots de passe est cochée dans les Propriétés de la base (cf. section **Sécurité des connexions**).

Note : Si le nom d'utilisateur envoyé par le navigateur existe dans 4D, pour des raisons de confidentialité le paramètre *$6* n'est alors pas rempli (il reçoit une chaîne vide).

- **Paramètre $0**
  La **Méthode base Sur authentification Web** retourne un booléen dans $0 :
  - Si $0 est **Vrai**, la connexion est acceptée.

- Si $0 est **Faux**, la connexion est refusée.

La **Méthode base Sur connexion Web** n'est exécutée que si la connexion est acceptée par **Sur authentification Web**.

ATTENTION : Si aucune valeur n'est passée dans *$0*, ou si *$0* n'est pas définie dans la **Méthode base Sur authentification Web**, la connexion sera considérée comme acceptée, et la **Méthode base Sur connexion Web** sera exécutée.

Notes :

- N'appelez aucun élément d'interface dans la **Méthode base Sur authentification Web** ((**ALERTE**, **DIALOGUE**, etc.), sinon son exécution sera interrompue et la connexion refusée. Il en est de même si une erreur se produit durant son traitement.
- Il est possible d'interdire l'exécution par *4DACTION* ou *4DSCRIPT* de chaque méthode projet à l'aide de l'option "Disponible via les balises HTML et URLs 4D (4DACTION...)" dans la boîte de dialogue des Propriétés des méthodes. Pour plus d'informations sur ce point, reportez-vous à la section **Sécurité des connexions**.

Appels de la Méthode base Sur authentification Web

La **Méthode base Sur authentification Web** est automatiquement appelée, quel que soit le mode, lorsqu'une requête ou un traitement nécessite l'exécution d'une méthode 4D. Elle est également appelée lorsque le serveur Web reçoit un URL statique invalide (par exemple, si la page statique demandée n'existe pas).

La **Méthode base Sur authentification Web** est donc appelée dans les cas suivants :

- lorsque 4D reçoit un URL débutant par *4DACTION/*
- lorsque 4D reçoit un URL débutant par *4DCGI/*
- lorsque 4D reçoit un URL débutant par *4DSYNC/*
- lorsque 4D reçoit un URL demandant une page statique inexistante
- lorsque 4D reçoit un URL d'accès à la racine et qu'aucune page d'accueil n'est définie dans les propriétés de la base ou via la commande **WEB FIXER PAGE ACCUEIL**
- lorsque 4D traite une balise *4DSCRIPT* dans une page semi-dynamique
- lorsque 4D traite une balise *4DLOOP* basée sur une méthode dans une page semi-dynamique

Note de compatibilité : La méthode base est également appelée lorsque 4D reçoit un URL débutant par *4DMETHOD/*. Cet URL obsolète est conservé par compatibilité uniquement.

A noter que la **Méthode base Sur authentification Web** n'est PAS appelée lorsque le serveur reçoit un URL demandant une page statique valide.

Exemple 1

Exemple de **Méthode base Sur authentification Web** en mode BASIC :

```
  `Méthode base Sur authentification Web
C_TEXTE($5;$6;$3;$4)
C_TEXTE($utilisateur;$motPasse;$IPBrowser;$IPServer)
C_BOOLEEN($utilisateur4D)
TABLEAU TEXTE($utilisateurs;0)
TABLEAU ENTIER LONG($nums;0)
C_ENTIER LONG($upos)
C_BOOLEEN($0)

$0:=Faux

$utilisateur:=$5
$motPasse:=$6
$IPBrowser:=$3
$IPServer:=$4

  `Pour des raisons de sécurité, refuser les noms qui contiennent @
Si(AvecJoker($utilisateur)|AvecJoker($motPasse))
   $0:=Faux
  `La méthode AvecJoker est décrite ci-dessous
Sinon
  `Vérifier si c'est un utilisateur 4D
   LIRE LISTE UTILISATEURS($utilisateurs;$nums)
   $upos:=Chercher dans tableau($utilisateurs;$utilisateur)
   Si($upos >0)
      $utilisateur4D:=Non(Utilisateur supprime($nums{$upos}))
   Sinon
      $utilisateur4D:=Faux
   Fin de si

   Si(Non($utilisateur4D))
  `Ce n'est pas un utilisateur défini dans 4D, chercher dans la table des utilisateurs Web
      CHERCHER([WebUsers];[WebUsers]User=$utilisateur;*)
      CHERCHER([WebUsers]; & [WebUsers]Password=$motPasse)
      $0:=(Enregistrements trouves([WebUsers])=1)
   Sinon
      $0:=Vrai
   Fin de si
Fin de si
  `Est-ce une connexion intranet ?
Si(Sous chaine($IPBrowser;1;7)#"192.100.")
   $0:=Faux
```

```
   Fin de si
```

Exemple 2

Exemple de méthode base Sur authentification Web en mode DIGEST :

```
    //Méthode base Sur authentification Web
C_TEXTE($1;$2;$5;$6;$3;$4)
C_TEXTE($utilisateur)
C_BOOLEEN($0)
$0:=Faux
$utilisateur:=$5
    //Pour des raisons de sécurité, refuser les noms qui contiennent @
Si(AvecJoker($utilisateur))
   $0:=Faux
    //La méthode AvecJoker est décrite ci-dessous
Sinon
   CHERCHER([WebUsers];[WebUsers]User=$utilisateur)
   Si(OK=1)
      $0:=WEB Valider digest($utilisateur;[WebUsers]Mdp)
   Sinon
      $0:=Faux //Utilisateur inexistant
   Fin de si
Fin de si
```

La Méthode projet ***AvecJoker*** est décrite ci-dessous :

```
    //Méthode projet AvecJoker
    //AvecJoker ( Chaine ) -> Booléen
    //AvecJoker ( Nom ) -> Contient un joker

C_ENTIER LONG($i)
C_BOOLEEN($0)
C_TEXTE($1)
$0:=Faux
Boucle($i;1;Longueur($1))
   Si(Code de caractere(Sous chaine($1;$i;1))=Code de caractere("@"))
      $0:=Vrai
   Fin de si
Fin de boucle
```

Méthode base Sur connexion Web

La **Méthode base Sur connexion Web** peut être appelée dans les cas suivants :

- le serveur Web reçoit une requête débutant par l'URL *4DCGI*.
- le serveur Web reçoit une requête invalide.

Pour plus d'informations, reportez-vous ci-dessous au paragraphe “Appels de la Méthode base Sur connexion Web”.

Note de compatibilité : La méthode base est également appelée en cas de création d'un contexte en mode contextuel (mode obsolète pouvant être utilisé dans les bases 4D converties).

La requête doit auparavant avoir été "acceptée" par la **Méthode base Sur connexion Web** (si elle existe) et le serveur Web doit être lancé.

La **Méthode base Sur connexion Web** reçoit six paramètres de type Texte, passés par 4D ($1, $2, $3, $4, $5 et $6). Voici leur description :

Paramètres	Type	Description
$1	Texte	URL
$2	Texte	En-tête + corps HTTP (32 ko maximum)
$3	Texte	Adresse IP du navigateur
$4	Texte	Adresse IP appelée du serveur
$5	Texte	Nom d’utilisateur
$6	Texte	Mot de passe

Vous devez déclarer ces six paramètres de la manière suivante :

```
  ` Méthode base Sur connexion Web

C_TEXTE($1;$2;$3;$4;$5;$6)

  ` Code pour la méthode
```

- **Données supplémentaires de l’URL**
  Le premier paramètre (*$1*) est l’**URL** saisi par l'utilisateur dans la zone 'Adresse' de son navigateur Web, moins l’adresse hôte.
  Prenons l’exemple d’une connexion Intranet. Supposons que l’adresse **IP** de votre machine serveur Web 4D est *123.4.567.89*. Le tableau suivant liste les valeurs de *$1* selon l’**URL**saisi dans le navigateur Web :

URL saisi dans le navigateur	Valeur du paramètre $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients/Ajouter	/Clients/Ajouter
123.4.567.89/Faire_ceci/Si_OK/Faire_cela	/Faire_ceci/Si_OK/Faire_cela

  Notez que vous êtes libre d'utiliser ce paramètre à votre convenance. 4D ignore simplement les valeurs passées au-delà de la partie hôte de l’URL. Par exemple, vous pouvez établir une convention dans laquelle la valeur *"/Clients/Ajouter"* signifie “aller directement à l'ajout d'un nouvel enregistrement dans la table *[Clients]*.” En fournissant aux utilisateurs Web de votre base une liste des valeurs possibles et/ou des marqueurs par défaut, vous pouvez proposer des raccourcis vers différentes parties de votre application. Ainsi, les utilisateurs Web peuvent accéder rapidement aux ressources de votre site Web sans devoir naviguer dans toute la base à chaque connexion.

ATTENTION : Pour empêcher un utilisateur d'accéder directement à une base à l'aide d'un marqueur créé pendant une session précédente, 4D intercepte tout URL correspondant à un des URLs standard de 4D.

- **En-tête et corps de la requête HTTP**
  Le deuxième paramètre (*$2*) est l'en-tête suivi du corps de la requête **HTTP** envoyée par le navigateur Web. Notez que ces informations sont passées telles quelles à votre **Méthode base Sur connexion Web**. Le contenu varie en fonction du type de navigateur Web qui tente de se connecter.
  Avec Safari sous Mac OS, vous recevrez un en-tête semblable à celui-ci :

  ```
  GET /favicon.ico HTTP/1.1
  Referer: http://123.45.67.89/4dcgi/test
  User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; fr-fr) AppleWebKit/523.10.3 (KHTML, like Gecko)
  Version/3.0.4 Safari/523.10
  Cache-Control: max-age=0
  Accept: */*
  Accept-Language: fr-fr
  Accept-Encoding: gzip, deflate
  Connection: keep-alive
  Host: 123.45.67.89
  ```

  Avec Microsoft Internet Explorer 8 sous Windows, vous recevrez un en-tête semblable à celui-ci :

  ```
  GET / HTTP/1.1
  Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg,
  application/x-ms-xbap, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
  ```

```
Accept-Language: fr-FR
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET
CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C)
Accept-Encoding: gzip, deflate
Host: 123.45.67.89
Connection: Keep-Alive
```

Si votre application exploite ces informations, il est de votre ressort d'analyser l'en-tête et le corps.

Note : Pour des raisons de performances, la taille de ces données ne peut dépasser 32 Ko. Au-delà, elles sont tronquées par le serveur HTTP de 4D.

- **Adresse IP du navigateur**
  Le troisième paramètre (*$3*) reçoit l'adresse IP de la machine du navigateur. Cette information peut vous permettre, en particulier, de distinguer les connexions Intranet des connexions Internet.
  Note : 4D retourne les adresses IPv4 dans un format hybride IPv6 comprenant un préfixe de 96 bits, par exemple ::ffff:192.168.2.34 pour l'adresse IPv4 192.168.2.34. Pour plus d'informations, reportez-vous à la section **Prise en charge d'IP v6**.

- **Adresse IP demandée du serveur**
  Le quatrième paramètre (*$4*) reçoit l'adresse IP demandée du serveur Web 4D. En effet, 4D autorise le multi-homing, permettant d'exploiter des machines disposant de plusieurs adresses IP. Pour plus d'informations sur ce point, reportez-vous à la section **Paramétrages du serveur Web**.

- **Nom d'utilisateur et Mot de passe**
  Les paramètres *$5* et *$6* reçoivent le nom d'utilisateur et le mot de passe saisis par l'utilisateur dans la boîte de dialogue standard d'identification affichée par le navigateur, le cas échéant.
  Cette boîte de dialogue apparaît pour chaque connexion dès que l'option **Utiliser les mots de passe** est cochée dans les Propriétés de la base (cf. section **Sécurité des connexions**).

Note : Si le nom d'utilisateur envoyé par le navigateur existe dans 4D, pour des raisons de confidentialité le paramètre *$6* n'est alors pas rempli (il reçoit une chaîne vide).

Appels de la Méthode base Sur connexion Web

La **Méthode base Sur connexion Web** peut être utilisée comme point d'entrée dans le serveur Web 4D, soit à l'aide de l'URL spécial *4DCGI*, soit à l'aide d'URLs de commande personnalisés.

Attention : L'appel d'une commande 4D affichant un élément d'interface (**DIALOGUE**, **ALERTE**...) entraîne l'arrêt du traitement de la méthode.

La **Méthode base Sur connexion Web** est donc appelée dans les cas suivants :

- lorsque 4D reçoit l'URL */4DCGI*. La méthode base est appelée avec l'URL */4DCGI/<action>* dans *$1*.
- lorsqu'une page Web appelée avec un URL du type *<chemin>/<fichier>* n'est pas trouvée. La méthode base est appelée avec l'URL (*).
- lorsqu'une page Web est appelée avec un URL du type *<chemin>/* et qu'aucune page d'accueil par défaut n'est définie. La méthode base est appelée avec l'URL (*).

(*) Dans ces cas particuliers, l'URL reçu dans *$1* ne débute pas par le caractère "/".

Méthode base Sur fermeture process Web

La **Méthode base Sur fermeture process Web** est appelée par le serveur Web de 4D à chaque fois qu'une session Web est sur le point d'être refermée. Une session peut être refermée dans les cas suivants :

- lorsque le nombre maximum de sessions simultanées est atteint (100 par défaut, modifiable via la commande **WEB FIXER OPTION**), et que 4D a besoin d'en créer de nouvelles (4D détruit automatiquement le process de la session inactive la plus ancienne),
- lorsque la période maximale d'inactivité du process de la session est atteinte (480 minutes par défaut, modifiable via la commande **WEB FIXER OPTION**),
- lorsque la commande **WEB FERMER SESSION** est appelée.

Au moment de l'appel de la méthode base, le contexte de la session (variables et sélections générées par l'utilisateur) est toujours valide. Ce principe vous permet donc de stocker les données relatives à la session afin de pouvoir les réutiliser par la suite, en particulier via la **Méthode base Sur connexion Web**.

Note : Dans le contexte d'une session 4D Mobile (pouvant générer plusieurs process), la **Méthode base Sur fermeture process Web** est appelée pour chaque process Web refermé, vous permettant de sauvegarder tout type de donnée (variable, sélection, etc.) générée par le process de session 4D Mobile.

Un exemple d'utilisation de la **Méthode base Sur fermeture process Web** est fourni dans la section **Gestion des sessions Web**.

Gestion des sessions Web

Le serveur Web de 4D propose un mécanisme simple et complet de gestion des sessions utilisateur. Ce mécanisme automatique permet à des clients Web de réutiliser le même contexte (sélections et instances de variables) lors de requêtes successives.

Ce principe est basé sur l'utilisation d'un cookie privé posé par 4D lui-même : "4DSID". À chaque requête d'un client Web, 4D contrôle la présence et la valeur du cookie 4DSID :

- si le cookie a une valeur, 4D tente de retrouver le contexte d'origine du cookie parmi les contextes existants,
  - si le contexte est trouvé, il est réutilisé pour la requête, la méthode ***Compiler_Web*** n'est pas exécutée,
  - si aucun contexte n'est trouvé, 4D en crée un nouveau,
- si le cookie n'a pas de valeur ou s'il n'est pas présent (pour cause d'expiration par exemple), 4D crée un nouveau contexte.

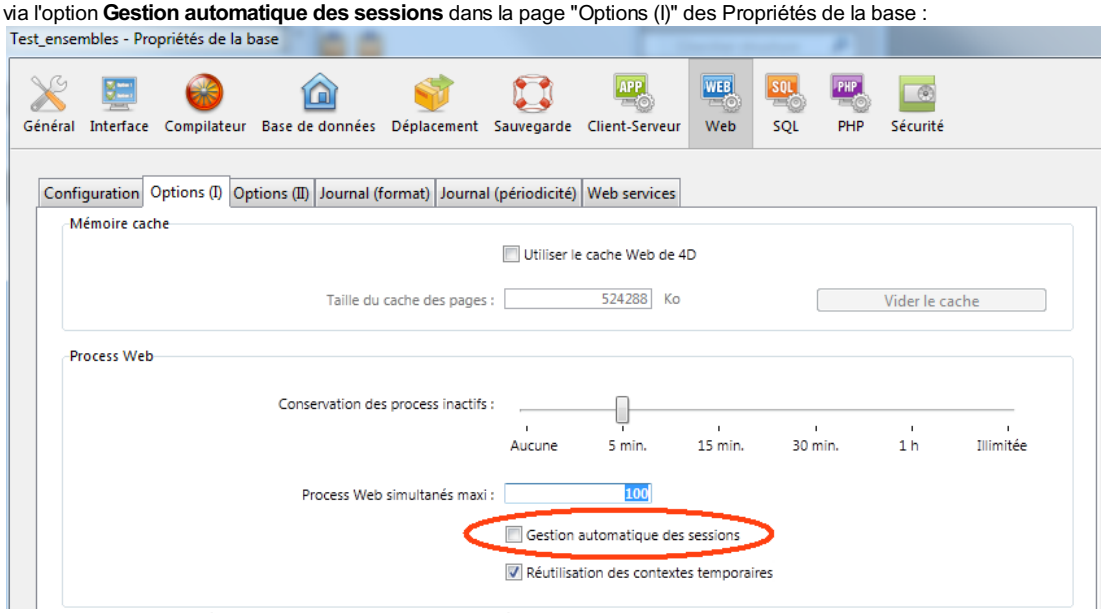
Activation et inactivation du mécanisme

Le mécanisme de gestion des sessions doit être activé sur votre serveur Web 4D pour que vous puissiez en bénéficier dans votre application.

Par défaut, ce mécanisme est activé dans les bases de données créées avec 4D v13 et suivantes. En revanche, pour des raisons de compatibilité, il est inactivé dans les bases de données converties depuis une version précédente de 4D. Vous devez l'activer explicitement pour pouvoir bénéficier de cette nouvelle fonctionnalité.

Vous pouvez gérer l'activation de la gestion automatique des sessions de deux manières :

- via l'option **Gestion automatique des sessions** dans la page "Options (I)" des Propriétés de la base :



  Dans ce cas, le paramétrage est permanent, il est stocké sur disque.
- via l'option Web conserver les sessions de la commande **WEB FIXER OPTION**. Dans ce cas, le paramétrage est défini pour la session uniquement et "surcharge" celui défini dans les propriétés de la base.

Dans les deux cas, le paramétrage est local au poste ; il peut donc différer entre le serveur Web de 4D Server et des serveurs Web 4D distants.

Expiration des cookies et conservation des contextes

La durée de vie du cookie en cas d'inactivité est de 8 heures par défaut (480 minutes) et est modifiable l'aide de la commande **WEB FIXER OPTION**. Il est possible de définir une durée de vie différente pour les cookies (option Web timeout session) et pour les process associés aux sessions sur le serveur (option Web timeout process) : vous pouvez souhaiter par exemple qu'un "panier" reste valide pendant 24 heures mais, pour des raisons d'optimisation, vous ne voulez pas maintenir de process aussi longtemps. Dans ce cas, vous pouvez fixer une durée de vie du process de 4 heures par exemple. A l'issue de ce délai, la **Méthode base Sur fermeture process Web** est appelée, vous permettant de stocker les variables et les sélections liées à la session, puis le process est tué. A la prochaine connexion du client Web (jusqu'à 24 heures plus tard), le cookie est renvoyé au serveur et vous pourrez recharger les informations de la session dans la **Méthode base Sur connexion Web** (cf. exemple ci-dessous).

Si nécessaire, vous pouvez forcer à tout moment l'expiration du cookie et donc clore la session à l'aide de la commande **WEB FERMER SESSION**.

Destruction des contextes inactifs

4D détruit automatiquement les plus anciens contextes inactifs lorsque le nombre maximum de contextes conservés est atteint (ce nombre de 100 par défaut peut être modifié à l'aide de la commande **WEB FIXER OPTION**).
Lorsqu'un contexte est sur le point d'être détruit (fermeture du process Web associé), la **Méthode base Sur fermeture process Web** est appelée, vous permettant de stocker les variables et les sélections du contexte, en prévision de réutilisations futures.

Exemple

Cet exemple illustre la simplicité de mise en oeuvre des sessions via les méthodes base Sur connexion Web et Sur fermeture session Web.

Voici le code de la **Méthode base Sur connexion Web** :

```
 // Sur connexion Web (ou Sur authentification Web)
C_TEXTE(www_SessionID)
Si(www_SessionID=WEB Lire ID session courante)
 // Compiler_Web n'a pas été appelé
 // Toutes les variables et les sélections existent
   ...
Sinon
 // Compiler_Web vient d'être exécuté
 // On est dans une nouvelle session, aucune variable ou sélection n'est définie.
 // On stocke l'id de la nouvelle session
   www_SessionID:=WEB Lire ID session courante

 // Initialisation de la session
 // Définition des sélections
 // Récupération de l'utilisateur sélectionné
   CHERCHER([User];[User]Login=www_Login)
   CHERCHER([prefs];[prefs]Login=www_Login)
 // Coordonnées de l'employé
   CHERCHER([emps];[emps]Name=[user]name)
   CHERCHER([company];[company]Name=[user]company)

 // Définition des variables
 // Lecture des préférences de cet utilisateur
   SELECTION VERS TABLEAU([prefs]name;prefNames;[prefs]values;prefValues)
   www_UserName:=[User]Name
   www_UserMail:=[User]mail

 // Fin d'initialisation de la session
Fin de si
```

Voici le code de la **Méthode base Sur fermeture process Web** :

```
 // Sur fermeture session Web
 // Après une période d'inactivité ou en cas de besoin, 4D ferme la session
C_TEXTE(www_SessionID)
www_SessionID:=""
 // On stocke des informations de la session
 // On enregistre les préférences de l'utilisateur précédemment connecté
CHERCHER([prefs];[prefs]Login=www_Login) // conservé dans la session
TABLEAU VERS SELECTION(prefNames;[prefs]name;prefValues;[prefs]values)

 // Important : le process est ensuite détruit
 // 4D efface les variables, les sélections, etc.
```

Cas du rejet des cookies

Le mécanisme de gestion des sessions étant basé sur l'utilisation de cookies, il ne sera pas possible au serveur HTTP de 4D de maintenir une session si le client Web rejette les cookies. Dans ce cas, chaque requête sera traitée comme une nouvelle connexion et provoquera l'exécution de la méthode ***Compiler_Web***.

Il est possible de vérifier que le client Web prend en charge les cookies à l'aide de la commande **WEB LIRE ENTETE HTTP**.

Sessions et gestion des adresses IP

Le serveur HTTP de 4D enregistre l'IP qui a débuté une session. Si un client Web situé à une adresse IP différente tente d'accéder à une session existante, l'erreur HTTP 400 est retournée au client.

Pages semi-dynamiques

Le serveur Web de 4D vous permet d'utiliser des **pages semi-dynamiques**.

Ces pages sont des 'templates' HTML contenant des **Balises de transformation 4D**, c'est-à-dire un mélange de code HTML statique et de références 4D ajoutées via les balises de transformation telles que 4DHTML, 4DIF, 4DINCLUDE. Ces balises sont insérées sous forme de commentaires type HTML (*<!--#LaBalise LeContenu-->*) dans le code HTML source.

Note : Une syntaxe alternative basée sur le $ est utilisable dans certaines conditions pour les balises 4DHTML, 4DTEXT et 4DEVAL afin de les rendre conformes au XML. Pour plus d'information, reportez-vous à la section **Syntaxe alternative pour 4DTEXT, 4DHTML, 4DEVAL**.

Au moment de leur envoi par le serveur HTTP, ces pages sont analysées et les balises qu'elles contiennent sont exécutées et remplacées par les données résultantes. Les pages reçues par les navigateurs sont alors la combinaison d'éléments statiques et de valeurs issues de 4D.

Principes

Vous pouvez donner par programmation des valeurs par défaut aux objets HTML en incluant par exemple *<!--#4DTEXT NomVar-->* dans le champ **valeur** de l'objet HTML ; **NomVar** est le nom de la variable process 4D telle qu'elle est définie dans le process Web courant — nom que vous mettez entre *<!—#—>*, c'est-à-dire la notation standard pour les commentaires HTML.

Note : Certains éditeurs HTML n'acceptent pas la saisie de la valeur *<!—#4DTEXT NomVar—>* dans le champ **valeur** des objets HTML. Dans ce cas, vous devrez la placer directement dans le code HTML.

En fait, la syntaxe *<!--#4DTEXT NomVar-->* permet d'insérer des données 4D partout dans la page HTML. Si, par exemple, vous écrivez :

<P>Bienvenue dans *<!--#4DTEXT vtSiteName-->*!</P>

La valeur de la variable 4D *vtSiteName* sera insérée dans la page HTML. Examinons un exemple :

```
 // Voici le code 4D assignant "4D4D" à la variable process vs4D
vs4D:="4D4D"
 //Puis envoyer la page HTML "AnyPage.HTM"
WEB ENVOYER FICHIER("AnyPage.HTM")
```

Le source de la page HTML **AnyPage.HTM** est le suivant :

```
<html> <head>     <title>AnyPage</title> <script language="JavaScript"><!-- function Is4DWebServer(){     return
(document.frm.vs4D.value=="4D4D") } function HandleButton(){     if(Is4DWebServer()){         alert("You are connected to 4D Web
Server!")     } else {         alert("You are NOT connected to 4D Web Server!") } //--></script> </head> <body> <form
action="/4DACTION/WWW_STD_FORM_POST" method="post" name="frm"> <p><input type="hidden" name="vs4D" value="<!--#4DTEXT vs4D-->"</p> <p>
<a href="JavaScript:HandleButton()"><img src="AnyGIF.GIF" border=0 align=bottom></a></p> <p><input type="submit" name="bOK"
value="OK"></p> </form> </body> </html>
```

La balise *<!--#4DTEXT -->* permet également d'insérer des **expressions 4D** dans les pages envoyées (champs, éléments de tableaux, etc.). Le principe de fonctionnement de la balise avec ce type de donnée est identique à celui des variables. Vous pouvez également insérer du code HTML dans des variables 4D à l'aide de la balise *4DHTML*. D'autres balises telles que *4DIF* permettent de contrôler le code exécuté. L'ensemble des balises utilisables est décrit dans la section **Balises de transformation 4D**.

Traitement des balises

L'analyse du contenu des pages semi-dynamiques envoyées par 4D s'effectue au moment de l'appel à **WEB ENVOYER FICHIER** (.htm, .html, .shtm, .shtml), **WEB ENVOYER BLOB** (blob de type text/html), **WEB ENVOYER TEXTE** et lors de l'envoi de pages appelées via des URLs. Dans ce dernier cas, à des fins d'optimisation, les pages suffixées ".htm" et ".html" ne sont PAS analysées. Pour "forcer" l'analyse des pages HTML dans ce cas, vous devez les suffixer ".shtm" ou ".shtml" (par exemple http://www.server.com/dir/page.shtm). Un exemple d'utilisation de ce type de page est fourni dans la description de la routine **WEB LIRE STATISTIQUES**. Les pages XML (.xml, .xsl) et les pages WML (.wml) sont également prises en charge et toujours analysées par 4D (cf. section **Support de XML et de WML**).

L'analyse peut également être effectuée en-dehors du contexte Web lorsque vous utilisez la commande **TRAITER BALISES 4D**.

En interne, l'analyseur travaille avec des chaînes utf-16, mais les données à analyser peuvent avoir été encodées différemment. Lorsque les balises contiennent du texte (par exemple 4DHTML), 4D convertit les données si nécessaire en fonction de leur provenance et des informations disponibles (*charset*). Voici un tableau récapitulatif des cas dans lesquels 4D analyse les balises contenues dans les pages HTML ainsi que les conversions éventuellement effectuées :

Action	Analyse du contenu des pages envoyées	Prise en charge de la syntaxe $ (*)	Jeu de caractères utilisé pour l'analyse des balises
Pages appelées par des URLs	X, sauf pages suffixées “.htm” ou “.html”	X, sauf pages suffixées “.htm” ou “.html”	Utilisation du charset passé en paramètre de l'en-tête "Content-Type" de la page. A défaut, recherche d'une balise META-HTTP EQUIV avec un charset. Sinon, utilisation du jeu de caractères par défaut du serveur HTTP
Commande **WEB ENVOYER FICHIER**	X	-	Utilisation du charset passé en paramètre de l'en-tête "Content-Type" de la page. A défaut, recherche d'une balise META-HTTP EQUIV avec un charset. Sinon, utilisation du jeu de caractères par défaut du serveur HTTP
Commande **WEB ENVOYER TEXTE**	X	-	Pas de conversion nécessaire
Commande **WEB ENVOYER BLOB**	X, si le BLOB est du type “text/html”	-	Utilisation du charset défini dans l'en-tête "Content-Type" de la réponse. Sinon, utilisation du jeu de caractères par défaut du serveur HTTP.
Inclusion par la balise *<!--#4DINCLUDE-->*	X	X	Utilisation du charset passé en paramètre de l'en-tête "Content-Type" de la page. A défaut, recherche d'une balise META-HTTP EQUIV avec un charset. Sinon, utilisation du jeu de caractères par défaut du serveur HTTP
Commande **TRAITER BALISES 4D**	X	X	Données Texte : pas de conversion. Données BLOB : conversion automatique depuis le jeu de caractères Mac-Roman par compatibilité

(*) La syntaxe alternative utilisant le $ est disponible pour les balises 4DHTML, 4DTEXT et 4DEVAL (cf. section **Syntaxe alternative pour 4DTEXT, 4DHTML, 4DEVAL**).

Encapsuler du JavaScript

Le code source JavaScript encapsulé dans les documents HTML est supporté par le serveur Web 4D, ainsi que l'insertion de fichiers JavaScript *.js* dans des documents HTML (par exemple *<SCRIPT SRC="...">*).

Avec **WEB ENVOYER FICHIER** ou **WEB ENVOYER BLOB**, vous envoyez une page que vous avez préparée avec un éditeur HTML ou construite avec 4D et sauvegardée comme document sur disque. Dans les deux cas, vous avez tout contrôlé sur la page et, par exemple, vous pouvez insérer des scripts JavaScript dans la section HEAD du document et utiliser des scripts avec une balise FORM. Ainsi, dans l'exemple ci-dessus, le script renvoie le formulaire "frm" car vous avez donné un nom au formulaire. Vous pouvez également déclencher, accepter ou rejeter la soumission du formulaire au niveau de la balise FORM.

Note : 4D supporte le transport d'Applets Java.

URLs et actions de formulaires

Le serveur Web 4D propose plusieurs URLs et actions de formulaires HTML spéciaux vous permettant d'effectuer différentes actions dans votre base de données.

Ces URLs sont les suivants :

- *4DACTION/*, permettant de lier un objet HTML à une méthode projet de votre base,
- *4DCGI/*, permettant d'appeler la **Méthode base Sur connexion Web** depuis tout objet HTML,
- *4DSYNC/*, permettant de synchroniser les données des tables.

En outre, le serveur Web 4D accepte plusieurs URLs supplémentaires :

- */4DSTATS*, */4DHTMLSTATS*, */4DCACHECLEAR* et */4DWEBTEST*, vous permettent d'obtenir diverses informations sur le fonctionnement de votre site Web 4D. Ces URLs sont décrits dans la section **Informations sur le site Web**.
- */4DWSDL*, permettant d'accéder au fichier de déclaration des Web Services publiés sur le serveur. Pour plus d'informations, reportez-vous à la section **Commandes du thème Web Services (Serveur)** et au manuel Mode Développement.

URL 4DACTION/

Syntaxe : *4DACTION/MaMéthode{/Param}*

Utilisation : URL ou Action de formulaire.

Cet URL vous permet de lier un objet HTML (texte, image, bouton...) à une méthode projet 4D. Le lien sera du type */4DACTION/MAMETH/PARAMS* où ***MAMETH*** est le nom de la méthode projet 4D à exécuter lorsque l'utilisateur clique sur le lien et *PARAMS* un paramètre optionnel de type Texte passé à la méthode dans *$1* (reportez-vous ci-dessous au paragraphe “Les paramètres Texte passés aux méthodes via des URLs”).
Lorsque 4D reçoit une requête */4DACTION/MAMETH/PARAMS*, la **Méthode base Sur authentification Web** (si elle existe) est appelée. Si elle retourne **Vrai**, la méthode ***MAMETH*** est exécutée.

4DACTION/ peut être associé à un URL dans une page Web statique. La syntaxe de l’URL sera de la forme suivante :

```
<A HREF="/4DACTION/MAMETH/PARAMS"> Faire Quelque Chose</A>
```

La méthode projet ***MAMETH*** doit généralement retourner une “réponse” (envoi de page HTML via **WEB ENVOYER FICHIER** ou **WEB ENVOYER BLOB**, etc.). Veillez à effectuer les traitements les plus courts possibles, afin de ne pas bloquer le navigateur.

Note : Une méthode appelée par *4DACTION* ne doit pas faire appel à des éléments d’interface (**DIALOGUE**, **ALERTE**...).

Attention : Pour qu’une méthode 4D puisse être exécutée via l’URL *4DACTION/*, elle doit disposer de l'attribut “Disponible via les balises HTML et les URLs 4D (4DACTION...)” (désélectionné par défaut), défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section **Sécurité des connexions**.

Exemple 1

Cet exemple décrit l'association de l'URL *4DACTION/* à un objet HTML image afin d'afficher dynamiquement une image dans la page. Vous insérez dans une page HTML statique les instructions suivantes :

```
<IMG SRC="/4DACTION/PICTFROMLIB/1000">
```

Le code de ***PICTFROMLIB*** est le suivant :

```
C_TEXTE($1) `Ce paramètre doit toujours être déclaré
C_IMAGE($VarImage)
C_BLOB($VarBlob)
C_ENTIER LONG($Numéro)
 //On récupère le numéro d’image dans la chaîne $1
$Numéro:=Num(Sous chaine($1;2;99))
LIRE IMAGE DANS BIBLIOTHEQUE($Numéro;$VarImage)
IMAGE VERS GIF($VarImage;$VarBlob)
WEB ENVOYER BLOB($VarBlob;"image/gif")
```

4DACTION pour poster des formulaires

Le serveur Web 4D permet également d'utiliser des formulaires “postés”, c’est-à-dire des pages HTML statiques renvoyant des données au serveur Web, et de récupérer facilement l'ensemble des valeurs. L’action du formulaire doit impérativement débuter par */4DACTION/NomMéthode*.

Note : Un formulaire peut être soumis dans deux modes (4D accepte les deux) :

- POST, généralement utilisé pour envoyer des données vers le serveur Web - dans une base de données.
- GET, généralement utilisé pour interroger le serveur Web - données en provenance de la base.

Lorsque le serveur Web reçoit un formulaire posté, il appelle la **Méthode base Sur authentification Web** (si elle existe). Si elle retourne **Vrai**, la méthode ***NomMéthode*** est exécutée. Dans cette méthode, vous devez appeler la commande **WEB LIRE VARIABLES** afin de récupérer le nom et la valeur de tous les champs inclus dans une page HTML soumise au serveur.

Note de compatibilité : Dans les bases converties, si l'option "Affectation automatique des variables" de la **Page Compatibilité** est cochée, la méthode projet spéciale ***COMPILER_WEB*** (si elle existe) est appelée au préalable ; 4D récupère les valeurs des champs HTML présents dans le formulaire et remplit automatiquement les variables 4D dans la méthode appelée avec leur contenu si elles portent le même nom. Ce fonctionnement est obsolète. Pour

plus d'informations, reportez-vous à la section **Traiter les données reçues**.

La syntaxe HTML à appliquer dans le formulaire est du type suivant :

- pour la définition de l'action du formulaire :

```
<FORM ACTION="/4DACTION/NomMéthode" METHOD=POST>
```

- pour la définition d'un champ du formulaire :

```
<INPUT TYPE=Type de champ NAME=nom du champ VALUE="Valeur par défaut">
```

Pour chaque champ du formulaire, 4D affecte la valeur du champ à la variable de même nom.

Exemple 2

Dans une base Web 4D, nous souhaitons que les navigateurs puissent effectuer des recherches parmi les enregistrements par l'intermédiaire d'une page HTML statique. Cette page s'intitule "search.htm". La base contient d'autres pages statiques, permettant par exemple d'afficher le résultat de la recherche ("results.htm"). Le type POST a été associé à la page, ainsi que l'action */4DACTION/SEARCH*. Voici le code HTML correspondant à cette page :

```
<FORM ACTION="/4DACTION/PROCESSFORM" METHOD=POST> <INPUT TYPE=TEXT NAME=VNAME VALUE=""><BR> <INPUT TYPE=CHECKBOX
NAME=vEXACT VALUE="Mot">Mot entier<BR> <INPUT TYPE=SUBMIT NAME=OK VALUE="Chercher"> </FORM>
```

En cours d'utilisation, vous tapez "ABCD" dans la zone de saisie, vous cochez l'option "Mot entier" et validez en cliquant sur **Chercher**.

Dans la requête envoyée au serveur Web :

```
VNAME="ABCD"
vEXACT="Mot"
OK="Chercher"
```

4D appelle la **Méthode base Sur authentification Web** (si elle existe), puis la méthode projet PROCESSFORM, dont voici le contenu :

```
C_TEXTE($1) //obligatoire pour le mode compilé
C_ENTIER LONG($vName)
C_TEXTE(vNAME;vLIST)
TABLEAU TEXTE($tabNoms;0)
TABLEAU TEXTE($tabVals;0)
WEB LIRE VARIABLES($tabNoms;$tabVals) //on récupère toutes les variables du formulaire
$vName:=Chercher dans tableau($tabNoms;"vNAME")
vNAME:=$tabVals{$vName}
Si(Chercher dans tableau($tabNoms;"vEXACT")=-1)  //Si l'option n'a pas été cochée
   vNAME:=VNAME+"@"
Fin de si
CHERCHER([Jockeys];[Jockeys]Nom=vNAME)
DEBUT SELECTION([Jockeys])
Tant que(Non(Fin de selection([Jockeys])))
   vLIST:=vLIST+[Jockeys]Nom+" "+[Jockeys]Tél+"<BR>"
   ENREGISTREMENT SUIVANT([Jockeys])
Fin tant que
WEB ENVOYER FICHIER("results.htm") //Envoi de la liste dans le
 //formulaire results.htm, qui contient une référence à la variable vLIST,
 //par exemple <!--4DHTML vLIST-->
 //...
Fin de si
```

URL 4DCGI/

Syntaxe : *4DCGI/<action>*

Utilisation : URL.

Lorsque le serveur Web 4D reçoit l'URL */4DCGI/<action>*, la **Méthode base Sur authentification Web** (si elle existe) est appelée. Si elle retourne **Vrai**, le serveur Web appelle la **Méthode base Sur connexion Web** en passant l'URL "tel quel" dans *$1*.

L'URL */4DCGI/* ne correspond à aucun fichier. Il a pour unique rôle d'appeler 4D via la **Méthode base Sur connexion Web**. Le paramètre "<action>" peut contenir n'importe quel type d'information.

Cet URL vous permet donc d'effectuer tout type de traitement. Il vous suffit de tester la valeur de *$1* dans la **Méthode base Sur connexion Web** ou dans une de ses sous-méthodes et d'effectuer dans 4D le traitement adéquat. Par exemple, vous pouvez construire des pages HTML statiques entièrement personnalisées d'ajout, de recherche ou de tri d'enregistrements, ou encore générer des images GIF à la volée. Des exemples d'utilisation de cet URL sont fournis dans les descriptions des commandes **IMAGE VERS GIF** et **WEB ENVOYER REDIRECTION HTTP**.

A l'issue du traitement, une "réponse" doit être retournée, à l'aide d'une des commandes d'envoi de données (**WEB ENVOYER FICHIER**, **WEB ENVOYER BLOB**, etc.).

ATTENTION : Veillez à effectuer les traitements les plus courts possibles, afin de ne pas bloquer le navigateur.

Paramètres texte passés aux méthodes 4D appelées via des URLs

4D envoie des paramètres Texte aux méthodes 4D appelées par des URLs spéciaux (*4DACTION/* et *4DCGI/*). Voici quelques remarques sur ces paramètres :

- Même si vous n'utilisez pas ces paramètres, vous devez les déclarer explicitement avec la commande **C_TEXTE**, sinon des erreurs runtime se produiront lorsque vous accèderez par le Web à une base exécutée en mode compilé. Le message est du type
  "Error in dynamic code
  Paramètres incorrects dans une commande EXECUTER
  Method Name:
  Line Number:
  Description: [<date et heure>]"
  Cette erreur runtime est provoquée par l'absence de déclaration du paramètre texte *$1* dans la méthode 4D appelée lorsque vous avez cliqué sur le lien HTML. Comme le contexte de l'exécution est la page HTML courante, l'erreur ne référence pas spécifiquement de ligne de méthode. Déclarer explicitement le paramètre texte *$1* permet de supprimer ces erreurs :

```
 //Méthode projet M_SEND_PAGE
C_TEXTE($1) // Ce paramètre DOIT être explicitement déclaré
 //...
WEB ENVOYER FICHIER($mapage)
```

- Le paramètre *$1* retourne des données supplémentaires placées à la fin de l'URL, et peut être utilisé pour passer des données de l'environnement HTML vers l'environnement 4D.

Paramètres à déclarer explicitement dans la méthode 4D appelée

Vous devez déclarer différents paramètres en fonction de l'origine et de la nature de l'appel de la méthode 4D.

- **Méthode base Sur authentification Web** (si elle existe) et **Méthode base Sur connexion Web**
  Vous devez déclarer les six paramètres de la connexion :

```
 ` Méthode base Sur connexion Web
C_TEXTE($1;$2;$3;$4;$5;$6) ` Ces paramètres DOIVENT être explicitement déclarés
```

- Méthode appelée par l'URL *4DACTION/*
  Vous devez déclarer le paramètre $1 :

```
 ` Méthode appelée par l'URL 4DACTION/
C_TEXTE($1) ` Ce paramètre DOIT être explicitement déclaré
```

- Méthode appelée par la balise *4DSCRIPT/*
  La méthode retourne une valeur dans $0. Vous devez déclarer les paramètres $0 et $1 :

```
 ` Méthode appelée par 4DSCRIPT/ sous forme de commentaire HTML
C_TEXTE($0;$1) ` Ces paramètres DOIVENT être explicitement déclarés
```

URL 4DSYNC/

Syntaxe :
4DSYNC/$catalog{/NomTable}
4DSYNC/NomTable{/NomTable}{/NomChamp1,...,NomChampN}{Params}

Utilisation : URL en méthode POST ou GET

Cet URL permet de synchroniser les données des tables de la base 4D locale avec une base distante via HTTP. Il peut par exemple être utilisé pour synchroniser une base 4D et une application cliente installée sur un Smartphone ou toute application HTTP tierce.

L'URL *4DSYNC/* peut être utilisé en méthode GET afin de récupérer les données de la base 4D ou en méthode POST afin de mettre à jour les données de la base 4D.

Voici les différentes requêtes HTTP utilisables :

- *GET /4DSYNC/$catalog*
  Retourne la liste des tables de la base et leur nombre d'enregistrements. Par exemple pour une structure avec deux tables PERSONS (2 enregistrements) et INVOICES (3 enregistrements), la syntaxe : *http://localhost/4DSYNC/$catalog/* retourne dans le navigateur: PERSONS 2 INVOICES 3

- *GET /4DSYNC/$catalog/NomTable*
  Retourne la description de la structure de *NomTable* (format XML).

- *GET /4DSYNC/NomTable/NomChamp1{,NomChamp2},...*
  Retourne les données des champs *NomChamp* de la table *NomTable*.

- *GET /4DSYNC/NomTable/NomChamp1?$stamp=0&$format=json*
  Retourne les données du champ *NomChamp* de la table *NomTable* à partir du *stamp* 0 et au format *json*.

- *POST /4DSYNC/NomTable/NomChamp1{,NomChamp2},...*
  Intègre dans la base 4D les modifications effectuées sur le client.

Note : Le format utilisé pour l'échange des données est le JSON (*JavaScript Object Notation*). La grammaire complète est disponible auprès des services techniques de 4D SAS.

Note : Pour que les mécanismes de synchronisation soient activés, l'option **Autoriser l'accès à la base de données via les URLs 4DSYNC** doit être cochée dans la page "Web/Configuration" des Propriétés de la base (cf. ci-dessous). Dans le cas contraire, les requêtes contenant l'URL 4DSYNC

échoueront.

Notes sur la synchronisation en HTTP

Vous devez tenir compte des principes suivants lorsque vous utilisez l'URL */4DSYNC/* :

- 4D travaille avec la structure virtuelle de la base si elle existe. Dans ce cas, les paramètres *NomTable* et *NomChamp* correspondent aux noms de tables et champs tels qu'ils ont été définis via les commandes **FIXER TITRES CHAMPS** et **FIXER TITRES TABLES**. A noter que la portée de ces commandes est la session, dans le cadre de 4D Server vous devez les appeler en procédure stockée sur le serveur.
- La réplication des champs de type Blob et Image n'est pas prise en charge.
- Pour que la synchronisation puisse être effectuée :
  - Le serveur HTTP doit être lancé.
  - L'option **Autoriser l'accès à la base de données via les URLs 4DSYNC** doit être cochée dans la page "Web/Configuration" des Propriétés de la base (cf. section **Sécurité des connexions**).
  - La propriété **Activer réplication** doit être cochée pour chaque table dont vous voulez synchroniser les données. Si une structure virtuelle a été définie, la table doit être incluse dans cette structure virtuelle. Attention, le fait de cocher cette option publie des informations supplémentaires, vous devez veiller à ce que les accès à votre base de données soient protégés (cf. description de l'option dans la section **Sécurité des connexions**).
- Lorsque 4D reçoit une requête */4DSYNC*, la **Méthode base Sur authentification Web** est appelée (hormis en cas de mot de passe incorrect, voir le schéma de connexion dans la section **Sécurité des connexions**). Si elle retourne **Vrai**, la requête est exécutée, sinon elle est rejetée.

Traiter les données reçues

Le serveur Web de 4D vous permet de récupérer les données 'postées', c'est-à-dire les données saisies par les utilisateurs via des formulaires Web et renvoyées au serveur via des boutons ou des éléments d'interface, en mode POST ou en mode GET.

Le serveur Web accepte plusieurs URLs spécifiques pouvant être associées aux boutons afin que l'envoi du formulaire déclenche le traitement côté serveur. Ces URLs sont décrits dans la section **URLs et actions de formulaires**.

Ce chapitre présente les principes à mettre en oeuvre afin de récupérer les données présentes dans les formulaires ayant été retournés au serveur.

Recevoir des valeurs dynamiques

Lorsque le serveur Web reçoit un formulaire "posté", vous pouvez récupérer dans 4D les valeurs des objets HTML qu'il contient. Ce principe peut être mis en oeuvre dans le cas d'un formulaire Web, envoyé par exemple avec **WEB ENVOYER FICHIER** ou **WEB ENVOYER BLOB**, dans lequel l'utilisateur saisit ou modifie des valeurs puis clique sur le bouton de validation. Dans ce cas, 4D peut récupérer les valeurs des objets HTML présents dans la requête de deux manières :

- à l'aide de la commande **WEB LIRE VARIABLES**,
- à l'aide des commandes **WEB LIRE PARTIE CORPS** et **WEB Lire nombre parties corps**.

La commande **WEB LIRE VARIABLES** permet de récupérer les valeurs sous forme de textes, tandis que les commandes **WEB LIRE PARTIE CORPS** et **WEB Lire nombre parties corps** permettent notamment de récupérer des fichiers postés, via des BLOBs.

Note de compatibilité (4D v13.4) : Dans les versions précédentes, 4D recopiait directement la valeur des variables reçues via un formulaire Web posté ou un URL GET, dans des variables process 4D (en mode compilé, ces variables devaient être déclarées dans la méthode COMPILER_WEB). Ce fonctionnement est supprimé à compter de 4D v13.4 ; il est maintenu par compatibilité dans les bases de données converties mais peut être désactivé à l'aide de l'option de compatibilité **Affectation automatique des variables** dans la **Page Compatibilité** des Propriétés de la base (il est recommandé de désélectionner cette option et d'utiliser les commandes **WEB LIRE VARIABLES** ou **WEB LIRE PARTIE CORPS** dans vos bases).

Considérons cette page de code HTML :

```
<html> <head>   <title>Welcome</title>   <script language="JavaScript"><!-- function GetBrowserInformation(formObj){
formObj.vtNav_appName.value = navigator.appName formObj.vtNav_appVersion.value = navigator.appVersion formObj.vtNav_appCodeName.value
= navigator.appCodeName formObj.vtNav_userAgent.value = navigator.userAgent return true } function LogOn(formObj){
if(formObj.vtUserName.value!=""){ return true } else { alert("Enter your name, then try again.") return false } } //--></script>
</head> <body> <form action="/4DACTION/WWW_STD_FORM_POST" method="post"  name="frmWelcome"  onsubmit="return
GetBrowserInformation(frmWelcome)">   <h1>Welcome to Spiders United</h1>   <p><b>Please enter your name:</b>   <input
name="vtUserName" value="" size="30" type="text"></p>   <p> <input name="vsbLogOn" value="Log On" onclick="return LogOn(frmWelcome)"
type="submit"> <input name="vsbRegister" value="Register" type="submit"> <input name="vsbInformation" value="Information"
type="submit"></p> <p> <input name="vtNav_appName" value="" type="hidden"> <input name="vtNav_appVersion" value="" type="hidden">
<input name="vtNav_appCodeName" value="" type="hidden"> <input name="vtNav_userAgent" value="" type="hidden"></p> </form> </body>
</html>
```

Lorsque la page est envoyée au navigateur Web par 4D, elle se présente ainsi :

Voici les caractéristiques de cette page :

- Elle comporte trois boutons Submit : *vsbLogOn*, *vsbRegister* et *vsbInformation*.
- Le Submit du formulaire quand vous cliquez sur LogOn est d'abord traité par la fonction JavaScript **LogOn**. Si aucun nom n'a été saisi, le formulaire n'est pas envoyé à 4D et une alerte JavaScript est affichée.
- Le formulaire comporte une méthode 4D POST et un script Submit (**GetBrowserInformation**) qui copie les propriétés du navigateur dans quatre objets cachés dont les noms commencent par *vtNav_App*.
- Vous remarquez aussi l'objet *vtUserName*.

Examinons maintenant la méthode 4D (nommée ici ***WWW_STD_FORM_POST***) qui est appelée lorsque l'utilisateur clique sur l'un des boutons du formulaire HTML :

```
 // Récupération de la valeur des variables
TABLEAU TEXTE($tabNoms;0)
TABLEAU TEXTE($tabValeurs;0)
WEB LIRE VARIABLES($tabNoms;$tabValeurs)
C_TEXTE($user)

Au cas ou

      // Le bouton Log On a été cliqué
   :(Chercher dans tableau($tabNoms;"vsbLogOn")#-1)
      $user :=Chercher dans tableau($tabNoms;"vtUserName")
      CHERCHER([WWW Users];[WWW Users]UserName=$tabValeurs{$user})
      $0:=(Enregistrements trouves([WWW Users])>0)
      Si($0)
         WWW POST EVENT("Log On";WWW Log information)
           // La méthode WWW POST EVENT sauvegarde l'information dans une table de la base
      Sinon
```

```
            $0:=WWW Register
               // La méthode WWW Register permet à un nouvel utilisateur Web de s'enregistrer
         Fin de si

            // Le bouton Register a reçu un clic souris
      :(Chercher dans tableau($tabNoms;"vsbRegister")#-1)
         $0:=WWW Register

            // Le bouton Information a reçu un clic souris
      :(Chercher dans tableau($tabNoms;"vsbInformation")#-1)
         WEB ENVOYER FICHIER("userinfos.html")
   Fin de cas
```

Voici les caractéristiques de cette méthode :

- Les valeurs des variables *vtNav_appName*, *vtNav_appVersion*, *vtNav_appCodeName* et *vtNav_userAgent* (liées aux objets HTML de même nom) sont récupérées via la commande **WEB LIRE VARIABLES** à partir des objets HTML créés par le script JavaScript **GetBrowserInformation**.
- Parmi les variables *vsbLogOn*, *vsbRegister* et *vsbInformation* liées aux trois boutons Submit, seule celle qui correspond au bouton ayant reçu le clic sera récupérée par la commande **WEB LIRE VARIABLES**. Quand le submit est effectué par l'un de ces trois boutons, le navigateur retourne à 4D la valeur du bouton sur lequel on a cliqué. Ce principe vous indique sur quel bouton on a cliqué. Notez que les boutons 4D (dans un formulaire 4D) sont des variables numériques. En revanche, en HTML, tous les objets sont des objets texte.

Si vous utilisez un objet SELECT, c'est la valeur de l'élément sélectionné dans l'objet qui est retournée dans la commande **WEB LIRE VARIABLES**, et non la position de l'élément dans le tableau comme dans 4D.

Les valeurs retournées par **WEB LIRE VARIABLES** sont toujours de type texte.

Prise en charge du chunked transfer encoding

A compter de 4D v15 R3, le serveur Web intégré de 4D prend en charge le mécanisme d'encodage de transfert morcelé (*chunked transfer encoding*) pour les fichiers téléchargés vers le serveur (*upload*). L'encodage de transfert morcelé est un mécanisme de transmission des données spécifié dans le protocole HTTP/1.1. Il permet de commencer à transférer des données via une série de "chunks" (morceaux) sans qu'il soit nécessaire de connaître la taille finale des données.

Note : Le serveur Web 4D prend également en charge l'encodage de transfert morcelé pour le téléchargement des données du serveur vers les clients Web (cf. **WEB ENVOYER DONNEES**).

Pour plus d'informations sur l'implémentation côté client des transferts morcelés, veuillez vous reporter à la RFC7230 ou à la page qui lui est consacrée sur Wikipedia.

Méthode projet COMPILER_WEB

La méthode **COMPILER_WEB**, si elle existe, est systématiquement appelée lorsque le serveur HTTP reçoit une requête dynamique et appelle le moteur de 4D. C'est par exemple le cas lorsque le serveur Web 4D reçoit un formulaire posté ou un URL contenant l'action 4DCGI/. Cette méthode est destinée à contenir les directives de typage et/ou d'initialisation de variables utilisées lors des échanges Web. Elle sera utilisée par le compilateur en cas de compilation de la base. La méthode **COMPILER_WEB** est commune à toutes les pages Web. Par défaut, la méthode **COMPILER_WEB** n'existe pas. Vous devez la créer explicitement.

Web Services : La méthode projet **COMPILER_WEB** est également appelée, si elle existe, à chaque requête SOAP acceptée. Vous devez utiliser cette méthode pour déclarer toutes les variables 4D associées à des arguments SOAP entrants et ce, pour toutes les méthodes publiées comme Web Services. En effet, l'utilisation de variables process dans les méthodes Web Services nécessite leur déclaration avant l'appel de la méthode. Pour plus d'informations sur ce point, reportez-vous à la description de la commande **SOAP DECLARATION**.

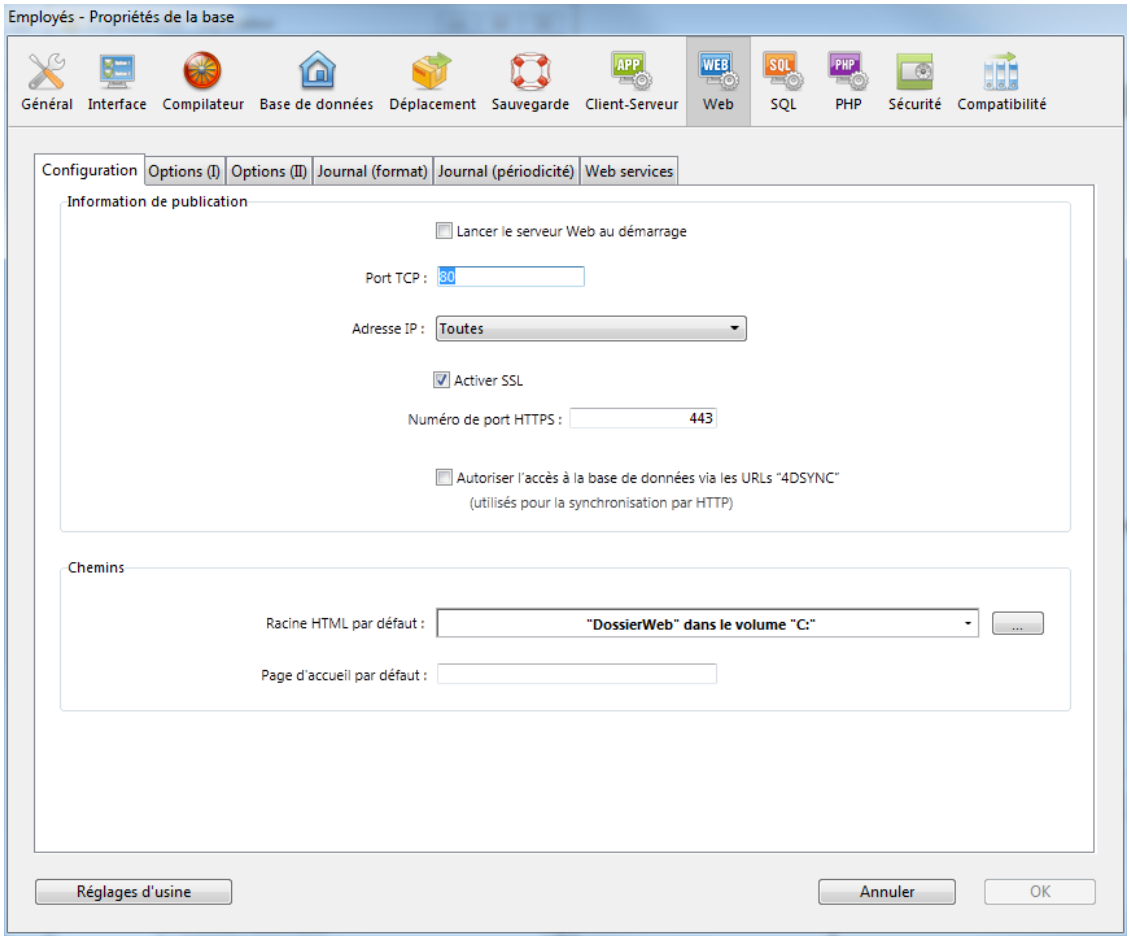
Paramétrages du serveur Web

Vous pouvez paramétrer le fonctionnement du serveur Web 4D à l'aide de paramètres définis dans la page **Web** des Propriétés de la base. Cette section décrit les paramètres des onglets **Configuration**, **Options (I)** et **(II)** de cette page.

- Les paramètres des pages **Journal** sont traités dans la section **Informations sur le site Web**.
- Les paramètres de la page **Web Services** sont traités dans le manuel "Mode Développement".

Note de compatibilité : Certains mécanismes Web présents dans les versions précédentes de 4D sont désormais considérés comme obsolètes (par exemple, "Supprimer le "/" sur les URLs inconnus). Par compatibilité, ces mécanismes restent utilisables dans les bases de données converties. Dans ce cas, vous pouvez les visualiser et éventuellement les désactiver dans la page **Compatibilité** des Propriétés de la base.

Page Configuration



Lancer le serveur Web au démarrage

Indique si le serveur Web doit être démarré dès le lancement de l'application 4D. Cette option est détaillée dans la section **Mise en route du serveur Web et gestion des connexions**.

Port TCP

Par défaut, 4D publie une base Web sur le port TCP standard du Web, qui est le port 80. Si ce port est déjà utilisé par un autre service Web, vous devez changer le port TCP utilisé par 4D pour votre base. La modification du port TCP permet également de lancer le serveur Web 4D sous Mac OS X sans être l'utilisateur root de la machine (cf. section **Mise en route du serveur Web et gestion des connexions**).
Pour cela, dans la zone de saisie “Port TCP”, indiquez le numéro de port TCP à utiliser pour cette base (c'est-à-dire un numéro de port TCP non utilisé par un autre service TCP/IP sur la machine).

Note : Si vous passez 0, 4D utilisera le numéro de port TCP par défaut, c'est-à-dire 80.

Au niveau du navigateur Web, vous devez inclure ce numéro de port TCP personnalisé dans la description de l'adresse utilisée pour vous connecter à la base Web. L'adresse doit être suivie du signe “deux-points” et du numéro de port. Si, par exemple, vous utilisez le port TCP numéro 8080, vous devrez spécifier dans le browser “123.45.67.89:8080”.

ATTENTION : Lorsque vous utilisez des numéros de port TCP autres que les numéros par défaut (80 pour le mode standard et 443 pour le mode SSL), prenez garde à ne pas spécifier de numéros de port qui se trouvent être les numéros par défaut d'autres services que vous employez simultanément. Si, par exemple, vous envisagez d'exploiter le protocole FTP sur la machine serveur Web, n'utilisez pas les numéros de ports TCP 20 et 21 qui sont les ports par défaut de ce protocole. Pour connaître les attributions standard des numéros de port TCP, vous pouvez vous reporter à la section **Annexe B, Numéros des ports TCP** dans la documentation de 4D Internet Commands. Les numéros de port inférieurs à 256 sont réservés pour les services standard et les numéros 256 à 1024 sont réservés pour les services spécifiques issus des plates-formes UNIX. Pour une sécurité maximum, il vous suffit de spécifier un numéro de port situé au-delà de ces intervalles, par exemple dans les 2000 ou 3000.

Adresse IP

Vous pouvez définir l'adresse IP sur laquelle le serveur Web doit recevoir les requêtes HTTP. Par défaut, le serveur répond sur toutes les adresses IP (option **Toutes**).

Note : A compter de 4D v14, le serveur HTTP prend automatiquement en charge la notation d'adresses IPv6 lorsque l'option **Toutes** est sélectionnée dans la liste "Adresse IP". Pour plus d'informations, reportez-vous à la section **Prise en charge d'IP v6**.

La liste déroulante "Adresse IP" liste automatiquement toutes les adresses IP présentes sur la machine. Lorsque vous sélectionnez une adresse particulière, le serveur ne répond qu'aux requêtes dirigées sur cette adresse.

Cette fonctionnalité est destinée aux serveurs Web 4D hébergés sur des machines ayant plusieurs adresses TCP/IP. C'est, par exemple, fréquemment le cas chez les fournisseurs d'hébergement Internet (MultiHoming). Vous pouvez donc placer plusieurs 4D ou 4D Server sur une même machine et réserver une adresse IP spécifique à chaque base publiée en Internet/Intranet.
La mise en place de cette fonctionnalité nécessite une configuration adéquate de la machine accueillant les différents serveurs Web :

- **Configuration MultiHoming sous Mac OS**

Pour mettre en place un système utilisant le MultiHoming sous Mac OS :

1. Ouvrez le tableau de bord TCP/IP.
2. Sélectionnez **manuel** dans le pop up menu **Configuration**.
3. Créez un fichier texte nommé "Secondary IP Addresses" et placez-le dans le sous-dossier "Préférences" de votre dossier Système.

Chaque ligne du fichier "Secondary IP Addresses" doit contenir une adresse IP secondaire ainsi que, si nécessaire, un masque de sous-réseau et une adresse de routeur.
Pour plus d'informations sur cette configuration, reportez-vous à la documentation fournie par Apple, Inc.

- **Configuration MultiHoming sous Windows**

Pour mettre en place un système utilisant le MultiHoming sous Windows :

1. Sélectionnez successivement les commandes suivantes (ou équivalentes en fonction de votre version de Windows) :
   Menu **Démarrer** > **Panneau de configuration** > **Connexions réseau et Internet** > **Connexions réseau** > **Connexion au réseau local** (Propriétés) > **Protocole Internet (TCP/IP)** > Bouton **Propriétés** > Bouton **Avancé...** La boîte de dialogue de configuration "Paramètres TCP/IP avancés" s'affiche.
2. Cliquez sur le bouton **Ajouter...** dans la zone "Adresses IP" et ajoutez les adresses IP supplémentaires.

Vous pouvez définir jusqu'à 5 adresses IP différentes. Pour cette opération, il se peut que vous ayez besoin de l'assistance d'un administrateur réseau.

Activer SSL

Indique si le serveur Web doit ou non accepter les connexions sécurisées. Cette option est décrite dans la section **Utiliser le protocole TLS**.

Numéro de port HTTPS

Permet de modifier le numéro du port TCP/IP utilisé par le serveur Web pour les connexions HTTP sécurisées via SSL (protocole HTTPS). Par défaut, le numéro du port HTTPS est 443 (valeur standard).
Vous pouvez souhaiter modifier ce numéro pour deux raisons principales :

- par souci de sécurité — en effet, les attaques des pirates contre les serveurs Web se concentrent généralement sur les ports TCP standard, c'est-à-dire 80 et 443.
- sous Mac OS X, pour permettre aux utilisateurs "standard" de lancer le serveur Web en mode sécurisé — en effet, sous Mac OS X, l'utilisation des ports TCP/IP réservés à la publication Web (0 à 1023) requiert des privilèges d'accès spécifiques : seul l'utilisateur "root" peut lancer une application utilisant ces ports. Pour que les utilisateurs autres que "root" puissent lancer le serveur Web, une solution consiste à modifier le numéro du port TCP/IP du serveur Web (cf. section **Mise en route du serveur Web et gestion des connexions**).
  Vous pouvez passer toute valeur valide (pour éviter les restrictions d'accès sous Mac OS X, il est nécessaire de passer une valeur supérieure à 1023). Pour plus d'informations sur les numéros de port TCP, reportez-vous ci-dessus au paragraphe "Port TCP".

Autoriser l'accès à la base de données via les URLs 4DSYNC

Cette option permet de contrôler la prise en charge des requêtes de synchronisation HTTP, comportant des URLs /4DSYNC. Cette option est décrite dans la section **Sécurité des connexions**.

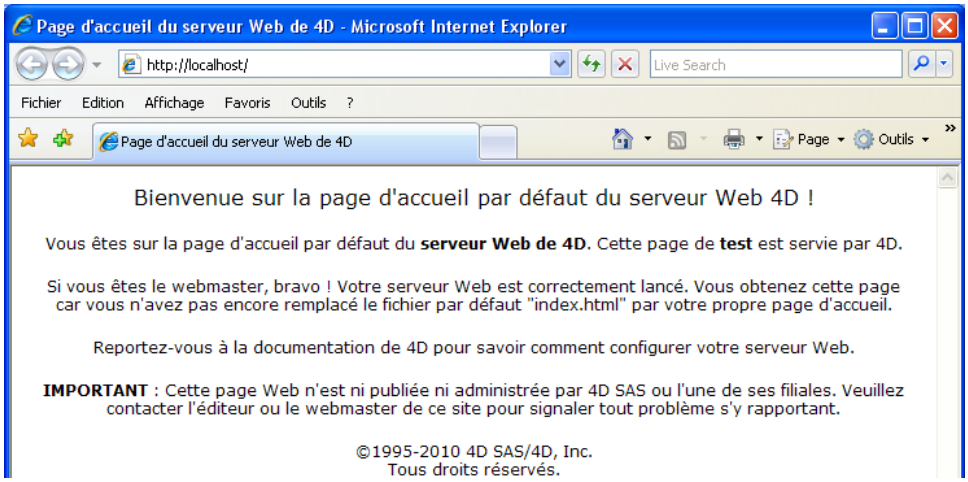
Racine HTML par défaut

Permet de définir l'emplacement par défaut des fichiers du site Web et indique le niveau hiérarchique sur le disque au-dessus duquel aucune requête ne pourra accéder. Cette option est décrite dans la section **Sécurité des connexions**.

Page d'accueil par défaut

Cette option permet de désigner la page d'accueil (page "Home") par défaut pour tous les navigateurs se connectant à la base. Cette page peut être statique ou semi-dynamique.

Par défaut, lors du premier démarrage du serveur Web, 4D crée une page d'accueil nommée "index.html" et la place dans le dossier racine HTML. Si vous ne modifiez pas ce paramétrage, tout navigateur se connectant au serveur Web obtient la page suivante :



Pour modifier la page d'accueil par défaut, vous pouvez simplement la remplacer par votre propre page "index.html" dans le dossier racine de la base ou saisir dans la zone le chemin d'accès relatif de la page que vous souhaitez définir. Le chemin d'accès doit être établi relativement au dossier racine HTML par défaut. Afin d'assurer la compatibilité multi-plate-forme de vos bases, le serveur Web 4D utilise, pour décrire les chemins d'accès, des conventions d'écriture particulières. Les règles de syntaxe sont les suivantes :

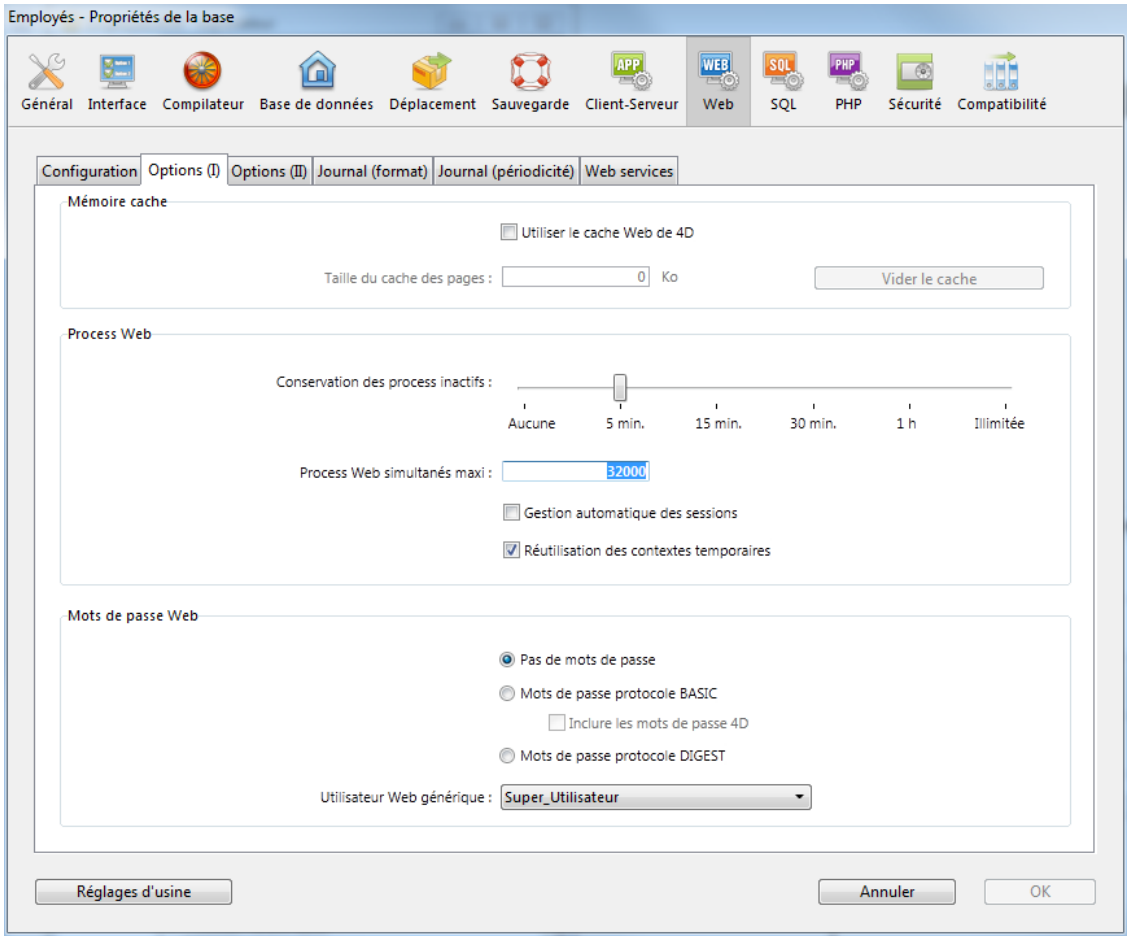
- les dossiers sont séparés par le caractère /
- le chemin ne doit pas se terminer par /
- pour "remonter" d'un niveau dans la hiérarchie des dossiers, saisissez ".." (point point) devant le nom du dossier
- le chemin ne doit pas commencer par /.

Par exemple, si vous souhaitez que la page d'accueil par défaut soit la page "MyHome.htm", placée dans le dossier "Web" (lui-même situé dans le dossier racine HTML par défaut de la base), saisissez **Web/MyHome.htm**

Note : Vous pouvez également définir une page d'accueil par défaut pour chaque process Web à l'aide de la routine **WEB FIXER PAGE ACCUEIL**.

Si vous ne spécifiez pas de page d'accueil par défaut, la **Méthode base Sur connexion Web** est appelée. Il vous revient alors de traiter la requête par programmation.

Page Options (I)



Mémoire cache

Le serveur Web 4D dispose d'un cache permettant de charger en mémoire les pages statiques, les images GIF, les images JPEG (<512 ko) et les feuilles de styles (fichiers .css), au fur et à mesure qu'elles sont demandées.
L'utilisation d'un cache permet d'augmenter de manière significative les performances du serveur Web en ce qui concerne l'envoi de pages statiques. Le cache est commun à tous les process Web.
Par défaut, le cache des pages statiques n'est pas activé. Pour l'activer, il suffit de cocher l'option **Utiliser le cache Web de 4D**.
Vous pouvez éventuellement modifier la taille du cache dans la zone **Taille du cache des pages**. La valeur à fixer dépend du nombre et de la taille des

pages statiques de votre site Web, ainsi que des ressources dont dispose la machine hôte.

Note : Au cours de l'utilisation de votre base Web, vous pourrez contrôler les performances du cache à l'aide de la routine **WEB LIRE STATISTIQUES**. Si par exemple vous constatez que le taux d'utilisation du cache est proche de 100%, vous pouvez envisager d'augmenter la taille qui lui est allouée.
Les URL particuliers */4DSTATS* et */4DHTMLSTATS* vous permettent également d'obtenir des informations sur l'état du cache. Reportez-vous à la section **Informations sur le site Web**.
Une fois le cache activé, lorsqu'une page est demandée par un navigateur, le serveur Web 4D la cherche d'abord dans le cache. Si elle s'y trouve, elle est immédiatement envoyée, sinon le programme charge la page depuis le disque et la place dans le cache. Lorsque le cache est plein et que de la place supplémentaire est requise, 4D "décharge" les pages les moins utilisées, par ordre d'ancienneté.

Vider le cache

Vous pouvez à tout moment vider le cache des pages et des images qu'il contient (par exemple si vous avez effectué des modifications sur une page statique et souhaitez qu'elle soit rechargée dans le cache). Pour cela, il vous suffit de cliquer sur le bouton **Vider le cache**. Le cache est alors immédiatement vidé.

Note : Vous pouvez également utiliser l'URL spécial **/4DCACHECLEAR**.

Conservation des process inactifs

Permet de définir le délai maximum avant fermeture (timeout) des process Web inactifs sur le serveur.

Process Web simultanés maxi

Cette option indique la limite strictement supérieure du nombre de process Web de tout type (process Web standard ou appartenant à la "réserve" de process) pouvant être simultanément ouverts sur le serveur. Ce paramètre permet de prévenir la saturation du serveur Web 4D pouvant se produire lors d'un envoi massif de requêtes.
Par défaut, le nombre est de 32 000 (autrement dit, jusqu'à 31 999 process Web peuvent être créés simultanément). Vous pouvez passer toute valeur incluse entre 10 et 32 000.

Lorsque ce nombre maximum (moins un) de process Web concurrents est atteint, 4D ne crée plus de nouveau process et retourne le message "Serveur non disponible" (statut HTTP 503 - Service Unavailable) à toute nouvelle requête.

Note : Le nombre maximum de process Web peut également être défini à l'aide de la commande **WEB FIXER OPTION**.

A propos de la réserve de process Web

La "réserve" de process Web permet d'augmenter la réactivité du serveur Web. Cette réserve est dimensionnée par un minimum (0 par défaut) et un maximum (10 par défaut) de process à recycler. Ces valeurs peuvent être modifiées à l'aide de la commande **FIXER PARAMETRE BASE**. Lors du changement du nombre maximum de process Web, si celui-ci est inférieur à la limite supérieure de la "réserve", cette limite est alors abaissée au nombre maximum de process Web.

Comment déterminer la valeur à passer ?

En théorie, le nombre maximum de process Web est le résultat de la division Mémoire disponible / Taille de la pile d'un process Web(*). Une autre solution consiste à visualiser les informations sur les process Web affichées dans l'Explorateur d'exécution : le nombre courant de process Web et le nombre maximum atteint depuis le démarrage du serveur Web sont indiqués.

(*) La taille de la pile allouée par 4D pour un process Web est d'environ 512 Ko pour les versions 64 bits, et d'environ 256 Ko pour les versions 32 bits (valeurs indicatives, pouvant varier en fonction du contexte).

Gestion automatique des sessions

Permet d'activer ou d'inactiver le mécanisme interne de prise en charge automatique des sessions utilisateurs par le serveur HTTP de 4D. Ce mécanisme est détaillé dans la section **Gestion des sessions Web**.

Par défaut, ce mécanisme est activé dans les bases de données créées à compter de 4D v13. En revanche, pour des raisons de compatibilité, il est inactivé dans les bases de données créées avec une version antérieure de 4D. Vous devez l'activer explicitement pour pouvoir bénéficier de cette fonctionnalité.

Lorsque cette option est cochée, l'option "Réutilisation des contextes temporaires" est automatiquement cochée (et verrouillée).

Réutilisation des contextes temporaires (en mode distant)

Permet d'optimiser le fonctionnement du serveur Web de 4D en mode distant en recyclant les process Web créés pour le traitement de requêtes Web précédentes. En effet, le serveur Web de 4D a besoin d'un process Web spécifique pour le traitement de chaque requête Web ; en mode distant, lorsque cela s'avère nécessaire, ce process se connecte au poste 4D Server afin d'accéder aux données et au moteur de base de données. Il génère alors un contexte temporaire utilisant ses propres variables, sélections, etc. Une fois la requête traitée, le process est tué. Lorsque l'option **Réutilisation des contextes temporaires** est cochée, en mode distant 4D maintient les process Web spécifiques et les réutilise pour les requêtes suivantes. L'étape de création du process étant supprimée, les performances du serveur Web sont alors améliorées.
En contrepartie, vous devez veiller dans ce cas à initialiser systématiquement les variables utilisées dans les méthodes 4D afin de ne pas risquer d'obtenir des résultats erronés. De même, il est nécessaire d'effacer les sélections ou enregistrements courants éventuellement définis au cours de la requête précédente.

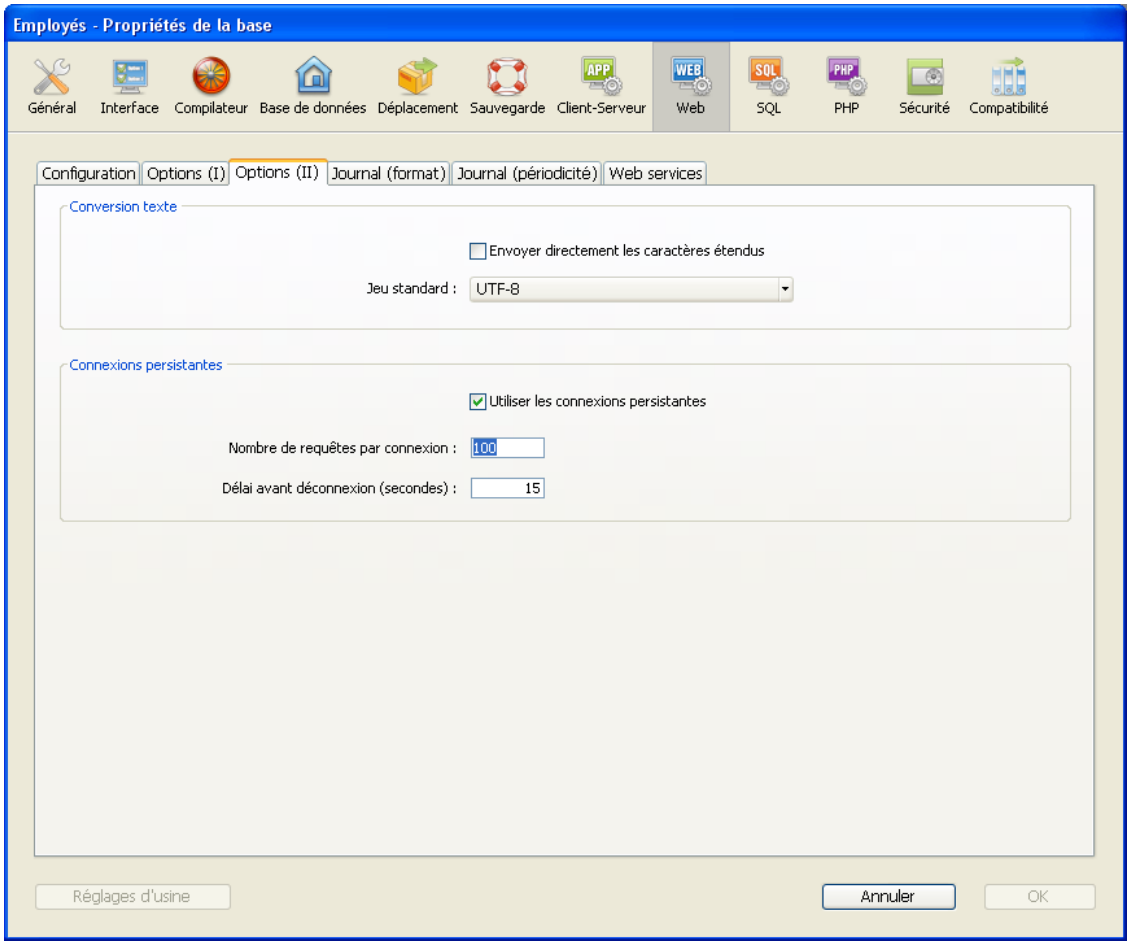
Notes :

- Cette option est automatiquement cochée (et verrouillée) lorsque l'option **Gestion automatique des sessions** est cochée. En effet, le mécanisme de gestion des sessions s'appuie sur le principe du recyclage des process Web : chaque session utilise un même process qui est maintenu durant toute la durée de vie de la session. A noter toutefois que les process de session ne sont pas "partageables" entre différentes sessions : une fois la session terminée, le process est automatiquement tué (il n'est pas réutilisé). Il n'est donc pas nécessaire de réinitialiser les sélections ou variables dans ce cas.
- Cette option n'a d'effet qu'avec un serveur Web 4D en mode distant. Avec 4D en mode local, les process Web autres que les process de session sont toujours tués après utilisation.

Zone "Mots de passe"

Paramétrage du système de protection des accès au site Web via des mots de passe. Cette option est décrite dans la section **Sécurité des connexions**.

Page Options (II)



Envoyer directement les caractères étendus

Par défaut, le serveur Web 4D convertit les caractères étendus présents dans les pages Web (dynamiques et statiques) au normes HTML avant de les envoyer. Ils sont ensuite interprétés par les navigateurs.
Vous pouvez paramétrer le serveur Web de manière à ce que les caractères étendus soient envoyés “tel quels”, sans conversion en entités HTML. Cette option permet un gain de vitesse important sur des systèmes étrangers (principalement japonais).
Pour cela, il suffit de cocher l'option **Envoyer directement les caractères étendus.**

Jeux standard

La liste déroulante **Jeu standard** vous permet de définir le jeu de caractères utilisé par le serveur Web 4D. Par défaut, le jeu de caractères est UTF-8.

Note : Ce paramétrage est également utilisé pour la génération des états rapides au format HTML (voir **Exécuter un état rapide**).

Connexions persistantes

Le serveur Web de 4D peut utiliser des connexions persistantes (keep-alive). L’option keep-alive permet de maintenir ouverte une seule connexion TCP pour l’ensemble des échanges effectués par un navigateur et le serveur afin d’économiser les ressources et d’optimiser les échanges.
L'option **Utiliser les connexions persistantes** permet d'activer ou d'inactiver les connexions TCP persistantes pour le serveur Web. Cette option est cochée par défaut. Dans la plupart des cas, il est conseillé de conserver cette option cochée car elle permet d’accélérer les échanges. Si le navigateur Web ne prend pas en charge les connexions keep-alive, le serveur Web 4D bascule automatiquement en HTTP/1.0.
La fonction keep-alive du serveur Web de 4D concerne toutes les connexions TCP/IP (HTTP, HTTPS). A noter toutefois que les connexions persistantes ne sont pas systématiquement utilisées pour tous les process Web 4D. Dans certains cas, d’autres mécanismes d’optimisation du serveur Web sont mis en oeuvre. Les connexions persistantes sont principalement efficaces lors de l’envoi de pages statiques.

Deux options permettent de régler le mécanisme des connexions persistantes :

- **Nombre de requêtes par connexion** : permet de définir le nombre maximum de requêtes pouvant transiter dans une même connexion persistante. Limiter le nombre de requêtes par connexion permet d’éviter les risques de saturation du serveur via l’envoi massif de requêtes (technique utilisée par les pirates).
  La valeur par défaut (100) peut être réduite ou augmentée en fonction des ressources de la machine hébergeant le serveur Web 4D.
- **Délai avant déconnexion** (timeout) : cette valeur définit le délai maximal (en secondes) pendant lequel le serveur Web maintient ouverte une connexion TCP tant qu’aucune nouvelle requête n’est reçue de la part du navigateur. A l’issue de ce délai, le serveur referme la connexion.
  Si le navigateur envoie une requête après la fermeture de la connexion, une nouvelle connexion TCP est automatiquement créée. Ce fonctionnement est transparent pour l’utilisateur.

Utiliser des process Web préemptifs

Le serveur Web intégré de 4D 64-bit pour Windows et pour OS X vous permet de tirer pleinement parti des ordinateurs multi-coeurs en utilisant des process Web préemptifs dans vos applications compilées. Vous pouvez configurer votre code lié au Web, y compris les balises HTML 4D et les méthodes base Web, afin qu'il s'exécute simultanément sur le plus grand nombre de coeurs possibles.

Pour plus d'informations sur la fonctionnalité des process préemptifs dans 4D, veuillez vous référer à la section **Process 4D préemptifs**.

Disponibilité du mode préemptif pour les pocess Web

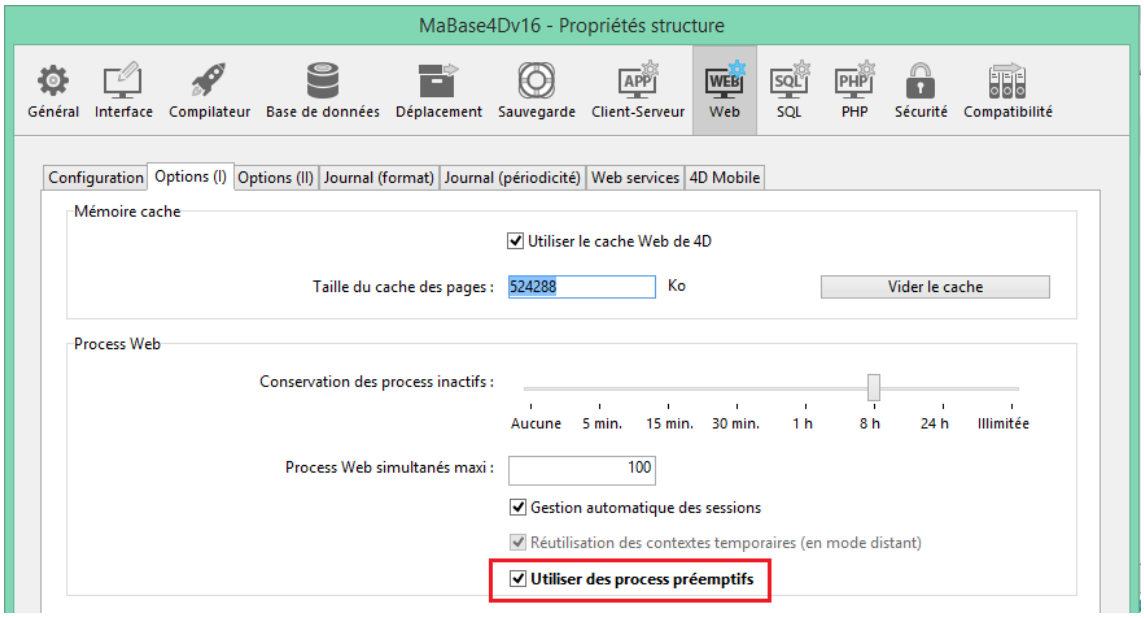
L'utilisation du mode préemptif pour les process Web est disponible seulement si les conditions suivantes sont réunies :

- utiliser une version 64-bit de 4D
- utiliser 4D Server ou 4D en mode local (le mode préemptif n'est pas pris en charge par 4D en mode distant)
- utiliser une base compilée
- cocher l'option **Utiliser des process préemptifs** dans les Propriétés de la base (voir ci-dessous)
- toutes les méthodes base et méthodes projet relatives au Web sont confirmées **thread-safe** par 4D Compiler

Si l'une de ces conditions n'est pas remplie, le serveur Web utilisera des process en mode coopératif.

Déclaration du mode préemptif pour le Serveur Web

Pour activer le mode préemptif pour le code concernant le Serveur Web de votre application, vous devez cocher l'option **Utiliser des process préemptifs** sur la page "Web/Option (I)" du dialogue des **Propriétés** de la base :



Lorsque cette option est cochée, le compilateur de 4D évaluera automatiquement la propriété **thread-safe** pour chaque partie du code relatif au serveur Web (voir ci-dessous) et retournera des erreurs en cas d'incompatibilité.

Ecrire du codeserveur Web thread-safe

Tout le code 4D exécuté par le serveur Web doit être **thread-safe** si vous souhaitez que les process Web soient lancés en mode préemptif. Lorsque l'option **Utiliser des process préemptifs** est cochée dans le dialogue des Propriétés de la base, les parties de l'application listées ci-dessous sont automatiquement évaluées par 4D Compiler :

- toutes les méthodes base relatives au Web :
  - **Méthode base Sur authentification Web**
  - **Méthode base Sur connexion Web**
  - **Méthode base Sur fermeture process Web**
  - **Méthode base Sur authentification 4D Mobile**
- la méthode projet ***compiler_web*** (quelle que soit l'option au niveau de sa propriété "Mode d'exécution")
- tout code traité par la commande **TRAITER BALISES 4D** en contexte Web, par exemple via des pages .shtml
- toute méthode projet avec l'attribut **Disponible via Balises HTML et URLs 4D (4DACTION...)**
- les triggers des tables ayant l'attribut **Exposer avec le service 4D Mobile**
- les méthodes projet disponibles via 4D Mobile (propriété "4D Mobile" cochée)

Pour chacune de ces méthodes ou parties de code, le compilateur vérifiera si les règles thread-safe sont respectées, et retournera une erreur en cas de problème. Pour plus d'informations à propos des règles thread-safe, veuillez vous référer au paragraphe **Ecrire une méthode thread-safe**.

Commandes 4D thread-safe et URLs Web

A partir de 4D v16, la plupart des commandes de 4D liées au Web, les méthodes et les URL de la base de données sont thread-safe et peuvent être

utilisées en mode préemptif.

Commandes 4D

Toutes les commandes 4D relatives au Web sont thread-safe, c'est-à-dire :

- toutes les commandes du thème "**Serveur Web**",
- toutes les commandes du thème "**Client HTTP** ".

Méthodes base

Les Méthodes base ci-dessous sont thread-safe et peuvent être utilisées en mode préemptif :

- **Méthode base Sur authentification Web**
- **Méthode base Sur connexion Web**
- **Méthode base Sur fermeture process Web**
- **Méthode base Sur authentification 4D Mobile**

Bien sûr, le code exécuté par ces méthodes doit aussi être thread-safe.

Web Server URLs

Les URLs Web Serveur ci-dessous sont thread-safe et peuvent être utilisées en mode préemptif :

- 4daction/ (la Méthode projet appelée doit aussi être thread-safe)
- 4dcgi/ (la Méthode base appelée doit aussi être thread-safe)
- 4dscript/ (obsolète en tant qu'URL, utilisé en tant que balise)
- 4dwebtest/
- 4dblank/
- 4dstats/
- 4dhtmlstats/
- 4dcacheclear/
- rest/
- 4dimgfield/ (généré par **TRAITER BALISES 4D** pour des requêtes web sur des champs image)
- 4dimg/ (généré par **TRAITER BALISES 4D** pour des requêtes web sur des variables image)

Les URLs 4D Web Serveur ci-dessous ne sont pas thread-safe et ne supportent pas le mode préemptif :

- 4dsync
- 4dsqlauth (obsolète, utilisé pour Flex 1.1)

Icône de process Web préemptif

Dans l'Explorateur d'exécution et dans la fenêtre d'administration de 4D Server, une icone spécifique s'affiche pour les process Web préemptifs :

Type de process	Icone
Méthode Web (process préemptif)	

Informations sur le site Web

Vous pouvez obtenir diverses informations sur le fonctionnement de votre site Web 4D :

- Vous pouvez contrôler le site par l'intermédiaire d'URL particuliers (*/4DSTATS*, */4DHTMLSTATS*, */4DCACHECLEAR* et */4DWEBTEST*).
- Vous pouvez générer un historique des requêtes.
- Vous pouvez visualiser la charge du serveur Web dans la page "Evaluation" de l'Explorateur d'exécution de 4D.

Note : Pour des raisons de sécurité, à compter de 4D v15 R2, la méthode HTTP TRACE est désactivée par défaut dans le serveur Web de 4D. Cela signifie que lorsqu'il reçoit une requête HTTP TRACE, le serveur Web de 4D retourne désormais une erreur 405 ("méthode non autorisée"). Si vous souhaitez autoriser explicitement la méhode HTTP TRACE, vous devez utiliser l'option Web TRACE HTTP avec la commande **WEB FIXER OPTION**.

URLs de gestion du serveur Web

Le serveur Web 4D accepte quatre URLs particuliers : */4DSTATS*, */4DHTMLSTATS*, */4DCACHECLEAR* et */4DWEBTEST*.

- */4DSTATS*, */4DHTMLSTATS* et */4DCACHECLEAR* sont accessibles uniquement au Super_Utilisateur et à l'Administrateur de la base. Si la base ne comporte pas de système de mots de passe 4D, ces URLs sont accessibles à tout utilisateur.
- */4DWEBTEST* est toujours accessible.

/4DSTATS

L'URL */4DSTATS* renvoie plusieurs informations sous la forme d'un tableau HTML (affichable dans un navigateur) :

- **Cache Current Size** : taille courante du cache du serveur Web (en octets)
- **Cache Max Size** : taille maximale du cache (en octets)
- **Cached Object Max Size** : taille maximum de chaque objet dans le cache (en octets)
- **Cache Use** : pourcentage du cache utilisé
- **Cached Objects** : nombre d'objets (pages, fichiers image...) présents dans le cache.

Pour plus d'informations sur le cache des pages statiques et des images, reportez-vous à la section **Paramétrages du serveur Web**.

Ces informations peuvent vous permettre de contrôler le fonctionnement de votre serveur et éventuellement d'adapter les paramètres correspondants.

Note : La commande **WEB LIRE STATISTIQUES** vous permet également d'obtenir des informations sur l'utilisation du cache pour les pages statiques.

/4DHTMLSTATS

L'URL */4DHTMLSTATS* renvoie, également sous forme de tableau HTML, les mêmes informations que l'URL */4DSTATS*, à la différence près que le champ **Cached Objects** ne dénombre que les pages HTML (sans les fichiers image). En outre, cet URL retourne le champ **Filtered Objects**.

- **Filtered Objects** : nombre d'objets du cache non comptabilisés par l'URL, notamment les images.

/4DCACHECLEAR

L'URL */4DCACHECLEAR* provoque l'effacement immédiat du cache des pages statiques et des images. Il permet donc de "forcer" la mise à jour de pages ayant été modifiées.

/4DWEBTEST

L'URL */4DWEBTEST* permet de contrôler le statut du serveur Web. Lorsque cet URL est appelé, 4D retourne un fichier texte contenant les champs HTTP suivants :

- **Date** : date du jour au format RFC 822
  Exemple : "Mon, 16 Jan 2012 13:12:50 GMT"
- **Server** : 4D_version/numéro de version interne
  Exemple : "4D_v12/12.3.0"
- **User-Agent** : nom et version @ adresse IP du client
  Exemple : "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:9.0.1) Gecko/20100101 Firefox/9.0.1 @ 127.0.0.1

Historique des requêtes

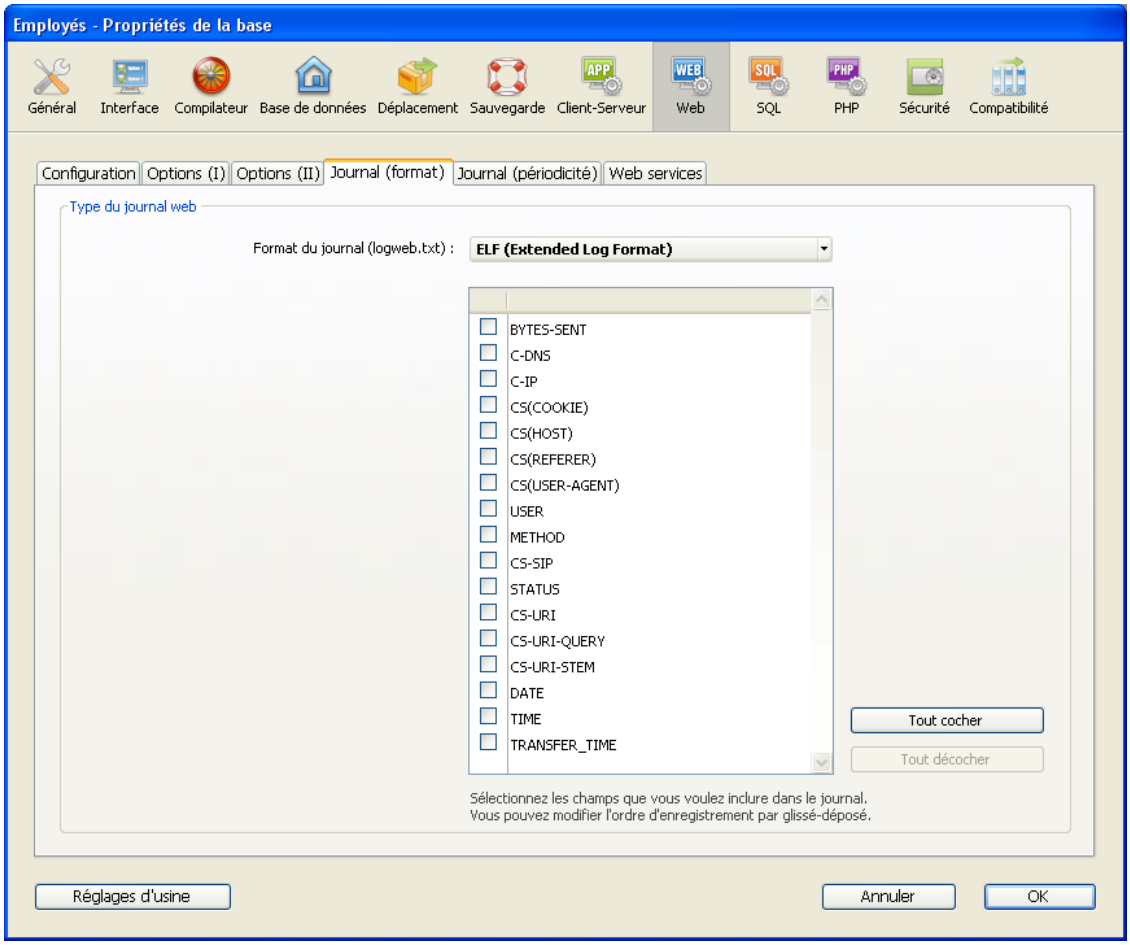
4D vous permet de générer un historique des requêtes.

Ce fichier est nommé "logweb.txt" et est automatiquement placé :

- avec 4D en mode local et 4D Server, dans le dossier Logs situé à côté du fichier de structure de la base.
- avec 4D en mode distant, dans le sous-dossier Logs du dossier base 4D client (dossier de cache).

Activation et format

L'activation et la configuration du contenu du fichier d'historique s'effectue dans les Propriétés de la base, page **Web/Journal (Format)** :



Note : L'activation et la désactivation du fichier d'historique des requêtes peut également être effectuée par programmation, à l'aide de la commande **FIXER PARAMETRE BASE** (4D v12) ou **WEB FIXER OPTION** (4D v13 et suivantes).

Le menu de format du journal propose les options suivantes :

- **Pas de journal** : lorsque cette option est sélectionnée, 4D ne génère pas d'historique des requêtes.
- **CLF (Common Log Format)** : lorsque cette option est sélectionnée, l'historique des requêtes est généré au format CLF. Avec le format CLF, chaque ligne du fichier représente une requête sous la forme :
  hôte rfc931 utilisateur [JJ/MMM/AAAA:HH:MM:SS] "requête" statut longueur
  Chaque champ est séparé par un espace, chaque ligne se termine par la séquence CR/LF (caractère 13, caractère 10).
  - hôte : adresse IP du client (ex. 192.100.100.10)
  - rfc931 : information non gérée par 4D, c'est toujours - (signe moins)
  - utilisateur : nom de l'utilisateur tel qu'il s'est authentifié, sinon - (signe moins). Si le nom de l'utilisateur contient des espaces, ils sont remplacés par des _ (tiret bas).
  - JJ : jour, MMM : mois abrégé en 3 lettres et toujours en anglais (Jan, Feb, ...), AAAA : année, HH : heure, MM : minutes, SS : secondes

  La date et l'heure sont locales au serveur
  - requête : requête envoyée par le client (ex. GET /index.htm HTTP/1.0)
  - statut : réponse donnée par le serveur.
  - longueur : taille des données renvoyées (hors en-tête HTTP) ou 0.

  Les valeurs possibles de statut sont :
  200 : OK
  204 : Pas de contenu
  302 : Redirection
  400 : Mauvaise requête
  401 : Authentification requise
  404 : Non trouvé
  500 : Erreur interne
  Le format CLF ne peut pas être personnalisé.

- **DLF (Combined Log Format)** : lorsque cette option est sélectionnée, l'historique des requêtes est généré au format DLF. Le format DLF est semblable au format CLF dont il reprend exactement la structure. Il contient simplement deux champs HTTP supplémentaires à la fin de chaque requête : Referer et User-agent.
  - Referer : contient l'URL de la page pointant vers le document demandé.
  - User-agent : contient le nom et la version du navigateur ou du logiciel client à l'origine de la requête.

  Le format DLF ne peut pas être personnalisé.

- **ELF (Extended Log Format)** : lorsque cette option est sélectionnée, l'historique des requêtes est généré au format ELF. Le format ELF est largement répandu dans le monde des serveurs HTTP. Il permet de construire des historiques sophistiqués, répondant à des besoins spécifiques. Pour cette raison, le format ELF est personnalisable : il est possible de choisir les champs à enregistrer ainsi que leur ordre d'insertion dans le fichier.

- **WLF (WebStar Log Format)** : lorsque cette option est sélectionnée, l'historique des requêtes est généré au format WLF. Le format WLF a été développé spécifiquement pour le serveur 4D WebSTAR. Il est semblable au format ELF, il dispose simplement de champs supplémentaires. Comme le format ELF, il est personnalisable.

Configurer les champs

Lorsque vous choisissez le format ELF (Extended Log Format) ou WLF (WebStar Log Format), la zone "Formatage du journal" affiche les champs disponibles pour le format. Vous devez sélectionner chaque champ à inclure dans l'historique. Pour cela, utilisez les flèches de commande ou procédez par

glisser-déposer.

Note : Il n'est pas possible de sélectionner deux fois le même champ.

Le tableau suivant liste les champs disponibles pour chaque format (par ordre alphabétique) et décrit son contenu :

Champ	ELF	WLF	Valeur
BYTES_RECEIVED		X	Nombre d'octets reçus par le serveur
BYTES_SENT	X	X	Nombre d'octets envoyés par le serveur au client
C_DNS	X	X	Adresse IP du DNS (ELF : champ identique au champ C_IP)
C_IP	X	X	Adresse IP du client (par exemple 192.100.100.10)
CONNECTION_ID		X	Numéro unique de la connexion
CS(COOKIE)	X	X	Informations sur les cookies contenus dans la requête HTTP
CS(HOST)	X	X	Champ Host de la requête HTTP
CS(REFERER)	X	X	URL de la page pointant vers le document demandé
CS(USER_AGENT)	X	X	Informations sur le logiciel et le système d'exploitation du client
CS_SIP	X	X	Adresse IP du serveur
CS_URI	X	X	URI sur lequel la requête est effectuée
CS_URI_QUERY	X	X	Paramètres d'interrogation de la requête
CS_URI_STEM	X	X	Partie de la requête sans les paramètres d'interrogation
DATE	X	X	DD: jour, MMM: abréviation de 3 lettres pour le mois (Jan, Feb,...), YYYY: année
METHOD	X	X	Méthode HTTP utilisée pour la requête adressée au serveur
PATH_ARGS		X	Paramètres de la CGI : chaîne située après le caractère "$"
STATUS	X	X	Réponse fournie par le serveur
TIME	X	X	HH: heure, MM: minutes, SS: secondes
TRANSFER_TIME	X	X	Délai ayant été nécessaire au serveur pour générer la réponse
USER	X	X	Nom d'utilisateur s'il s'est authentifié, sinon - (signe moins). Si le nom d'utilisateur contient des espaces, ils sont remplacés par des _ (traits de soulignement)
URL		X	URL demandé par le client

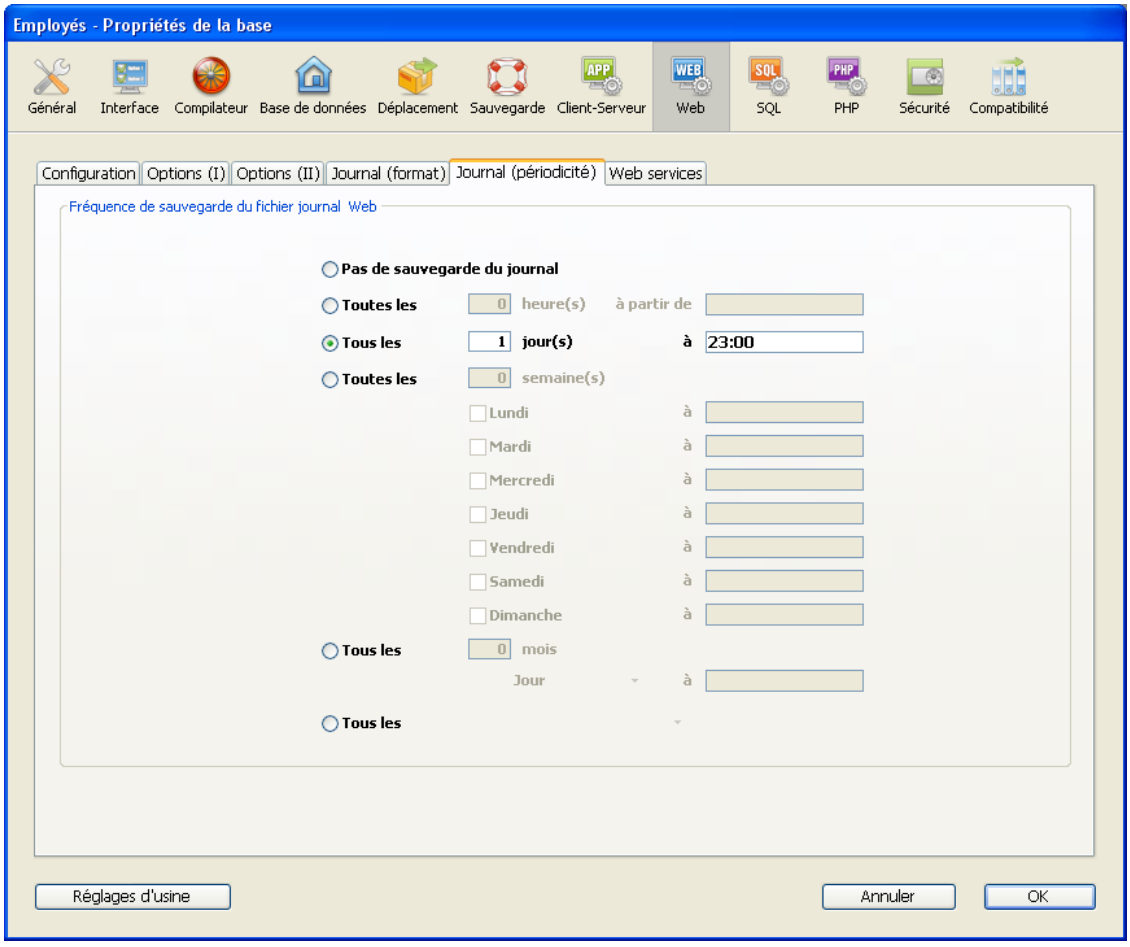
Note : La date et l'heure sont indiquées en GMT.

Périodicité de sauvegarde

Le fichier d'historique des requêtes Web pouvant atteindre une taille importante, il est possible de mettre en place un mécanisme d'archivage automatique. Le déclenchement de l'archivage peut être basé sur un délai (exprimé en heures, jours, semaines ou mois) ou une taille de fichier ; à chaque échéance, 4D referme et archive automatiquement le fichier d'historique courant et en crée un nouveau.
Lorsque l'archivage du ficher d'historique se déclenche, le fichier d'historique est archivé dans un dossier nommé "Logweb Archives", créé au même niveau que le fichier logweb.txt (c'est-à-dire à côté du fichier de structure de la base).
Le fichier archivé est renommé sur le modèle "DAAAA_MM_JJ_Thh_mm_ss.txt". Par exemple, pour un fichier archivé le 4 septembre 2006 à 15 heures 50 minutes et 7 secondes : "D2006_09_04_T15_50_07.txt"

Paramétrage des sauvegardes

Les paramètres d'archivage automatique de l'historique des requêtes sont définis dans la page **Web/Journal (Périodicité)** des Propriétés de la base :



Vous devez dans un premier temps choisir une échelle de fréquence (jours, semaines...) ou le critère de taille limite en cliquant sur le bouton radio

correspondant. Vous devez ensuite éventuellement préciser le moment de la sauvegarde.

- **Pas de sauvegarde du journal** : la fonction de sauvegarde périodique est inactivée.
- **Toutes les N heure(s)** : cette option permet de programmer la sauvegarde sur une base horaire. Vous pouvez saisir une valeur comprise entre 1 et 24.
  - **à partir de** : permet de définir l'heure à laquelle débutera la première sauvegarde horaire.
- **Tous les N jour(s) à N** : cette option permet de programmer la sauvegarde sur une base journalière. Saisissez 1 si vous souhaitez une sauvegarde quotidienne. Lorsque vous cochez cette option, vous devez indiquer l'heure à laquelle la sauvegarde doit être déclenchée.
- **Toutes les N semaine(s), jour à N** : cette option permet de programmer la sauvegarde sur une base hebdomadaire. Saisissez 1 si vous souhaitez une sauvegarde hebdomadaire. Lorsque vous cochez cette option, vous devez indiquer le ou les jour(s) de la semaine et l'heure à laquelle chaque sauvegarde doit être déclenchée. Vous pouvez cocher un ou plusieurs jour(s) de la semaine. Par exemple, vous pouvez utiliser cette option pour définir deux sauvegardes hebdomadaires : une le mercredi et une le vendredi.
- **Tous les N mois, Ne jour à N** : cette option permet de programmer la sauvegarde sur une base mensuelle. Saisissez 1 si vous souhaitez une sauvegarde mensuelle. Lorsque vous cochez cette option, vous devez indiquer le jour de chaque mois auquel la sauvegarde doit être déclenchée, ainsi que l'heure de déclenchement.
- **Tous les N Mo** : cette option permet de programmer la sauvegarde sur la base de la taille du fichier d'historique des requêtes courant. La sauvegarde est automatiquement déclenchée lorsque le fichier atteint la taille définie. Vous pouvez fixer une taille limite de 1, 10, 100 ou 1000 Mo.

Note : En cas de sauvegarde périodique, si le serveur Web n'était pas lancé au moment théorique de la sauvegarde, 4D considère au lancement suivant que la sauvegarde a échoué et applique les paramétrages adéquats, définis dans les Propriétés.

Informations fournies par l'Explorateur d'exécution de 4D

La page Evaluation (rubrique "Web") de l'Explorateur d'exécution affiche différentes informations concernant le fonctionnement du serveur Web 4D, notamment :

- **Occupation du cache Web** : indique le nombre de pages présentes dans le cache Web ainsi que son pourcentage d'utilisation. Cette information n'est disponible que si le serveur Web est actif et si la taille du cache est différente de 0.
- **Temps d'activité du serveur Web** : indique la durée de fonctionnement (au format heures:minutes:secondes) du serveur Web. Cette information n'est disponible que si le serveur Web est actif.
- **Nombre de requêtes http** : indique le nombre total de requêtes HTTP reçues depuis le démarrage du serveur Web, ainsi que le nombre instantané de requêtes par secondes (mesure prise entre deux mises à jour de l'Explorateur d'exécution). Cette information n'est disponible que si le serveur Web est actif.

Utiliser le protocole TLS

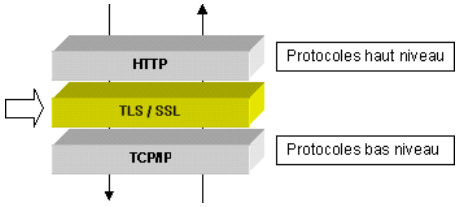
Le serveur Web 4D peut communiquer en mode sécurisé via le protocole TLS (Transport Layer Security) -- le successeur de SSL (Secured Socket Layer).

Qu'est-ce que le protocole TLS ?

Le protocole TLS (successeur de SSL) a pour but de sécuriser les communications entre deux applications — généralement un serveur Web et un navigateur. Ce protocole est largement répandu et compatible avec la plupart des navigateurs Web.

Au niveau de l'architecture réseau, le protocole de sécurité s'insère entre la couche TCP/IP (bas niveau) et le protocole de haut niveau HTTP, pour lequel il est principalement destiné.

Architecture réseau utilisant TLS :



Note : Le protocole TLS peut également être utilisé pour sécuriser les connexions client/serveur "classiques" de 4D Server ainsi que les connexions du serveur SQL. Pour plus d'informations, reportez-vous à la section **Crypter les connexions client/serveur** dans le manuel de référence de 4D Server ainsi qu'à la section **Configuration du serveur SQL de 4D** dans le manuel de référence SQL.

Le protocole TLS permet de garantir l'identité de l'émetteur et du récepteur, ainsi que la confidentialité et l'intégrité des informations échangées :

- **Identification des intervenants** : l'identité de l'émetteur et du récepteur sont confirmées.
- **Confidentialité des informations échangées** : les données envoyées sont cryptées afin de les rendre inintelligibles pour les tiers non autorisés.
- **Intégrité des informations échangées** : les données reçues n'ont pas été altérées, frauduleusement ou accidentellement.

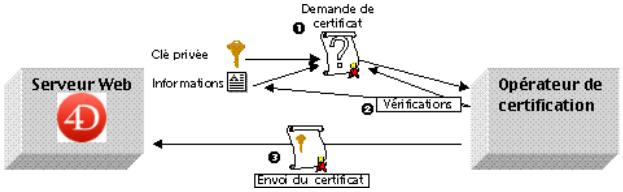
Les principes de sécurisation utilisés par TLS sont basés sur l'emploi d'un algorithme de cryptage utilisant une paire de clés : une clé privée et une clé publique.
La clé privée est utilisée pour crypter les données. Elle est conservée par l'émetteur (le site Web). La clé publique est utilisée pour décrypter les données. Elle est diffusée auprès des récepteurs (les navigateurs Web), via le certificat. L'emploi du TLS dans le cadre d'Internet requiert en effet l'entremise d'un opérateur de certification tel que, par exemple, Verisign®. Moyennant une participation financière du site Web demandeur, cet organisme délivre un certificat, garantissant l'identité du serveur et contenant la clé publique permettant la communication en mode sécurisé.

Note : Pour plus d'informations sur les principes généraux de cryptage et d'emploi de clés publiques/clés privées, reportez-vous à la description de la commande **CRYPTER BLOB**.

Obtenir un certificat numérique

La mise en place d'un serveur fonctionnant en TLS nécessite un certificat numérique délivré par un opérateur de certification. Ce certificat renferme diverses informations dont la carte d'identité du site ainsi que la clé publique utilisée pour communiquer avec lui. Il est transmis aux clients (navigateurs Web) se connectant au site. Une fois qu'il est accepté, la communication en mode sécurisé s'établit.

Note : Pour qu'un navigateur accepte les certificats d'une autorité de certification, celle-ci doit être répertoriée dans ses Propriétés.



Le choix de l'autorité de certification dépend de plusieurs facteurs. Plus l'autorité est "connue", plus le nombre de navigateurs acceptant les certificats qu'elle délivre sera important, mais plus le prix à payer sera élevé. Verisign® est une des autorités de certifications les plus utilisées.

Pour obtenir un certificat numérique :

1. Générez une "clé privée" à l'aide de la commande **GENERER CLES CRYPTAGE**.

Attention : Pour des raisons de sécurité, la clé privée ne doit JAMAIS être diffusée sur un réseau. En fait, elle ne doit pas quitter le poste serveur. Pour le serveur Web, le fichier **Key.pem** doit être placé dans le dossier de la structure de la base.

2. Etablissez une demande de certificat à l'aide de la commande **GENERER DEMANDE CERTIFICAT**.

3. Envoyez la demande de certificat à l'autorité de certification que vous avez choisie.
Pour remplir la demande de certificat, il vous sera peut-être nécessaire de contacter l'autorité de certification. Les autorités de certification vérifient la réalité des informations qui leur ont été transmises.
La demande de certificat est générée dans un BLOB au format PKCS encodé en base64 (format PEM). Ce principe autorise le copier-coller des clés sous forme de texte et leur envoi par E-mail en toute sécurité, sans risque d'altération de leur contenu. Vous pouvez donc par exemple sauvegarder le BLOB contenant la demande de certificat dans un document texte (à l'aide de **BLOB VERS DOCUMENT**), puis l'ouvrir et copier-coller son contenu dans un E-mail ou un formulaire Web destiné à l'autorité de certification.

4. Une fois que vous avez reçu votre certificat, créez un fichier texte que vous nommerez "Cert.pem" et copiez dans ce fichier le contenu du certificat.
Vous pouvez recevoir votre certificat sous plusieurs formes (généralement via un E-mail ou un formulaire HTML). 4D accepte la plupart des formats de texte (OS X, PC, Linux...) pour les certificats. En revanche, le certificat doit être au format PEM, c'est-à-dire PKCS encodé en base64.

Note : Les caractères de fins de ligne CR ne sont pas pris en charge (vous devez utiliser CRLF ou LF).

5. Placez le fichier "cert.pem" à l'emplacement adéquat. Pour le serveur Web, il s'agit du dossier contenant la structure de la base.
Le serveur Web peut dès lors fonctionner en mode sécurisé. La durée de validité d'un certificat varie généralement entre six mois et un an.

Installation et activation du TLS dans le serveur Web de 4D

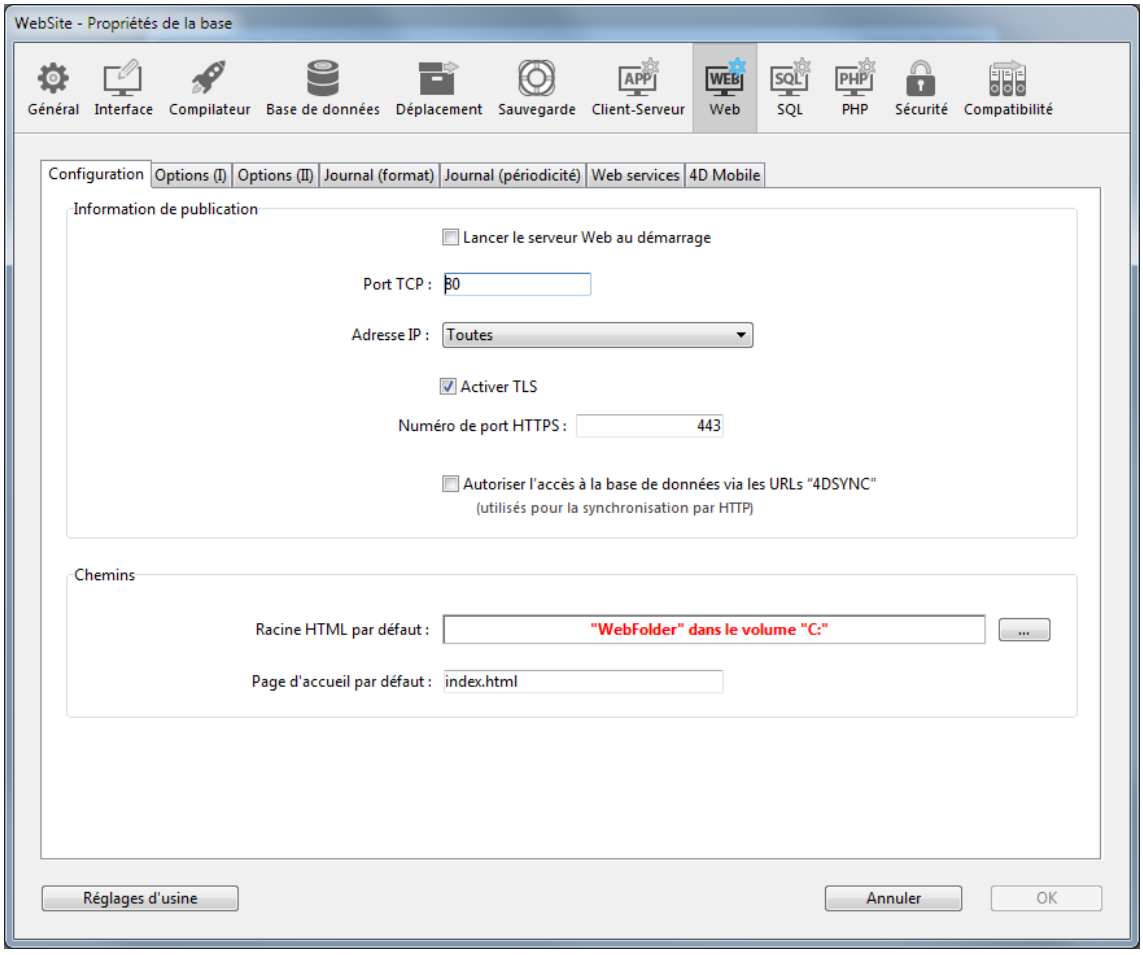
Pour que vous puissiez utiliser le protocole TLS avec le serveur Web de 4D, plusieurs éléments doivent être présents sur le serveur, à différents emplacements :

- *4DSLI.DLL* (Windows) ou *4DSLI.bundle* (Mac OS): interface de la couche sécurisée dédiée à la gestion du TLS.
  Ce fichier est installé par défaut, il est placé :
  - sous Windows, à côté du fichier exécutable de l'application 4D ou 4D Server
  - sous Mac OS, dans le sous-dossier *Native Components* du progiciel 4D ou 4D Server.

- *key.pem* (document contenant la clé de cryptage privée) et *cert.pem* (document contenant le certificat) :
  - avec 4D en mode local ou 4D Server, ces fichiers doivent se trouver à côté du fichier de structure de la base.
  - avec 4D en mode distant, ces fichiers doivent se trouver dans le dossier des ressources locales de la base sur le poste distant (pour plus d'informations sur l'emplacement de ce dossier, reportez-vous au paragraphe Dossier base 4D Client dans la description de la commande **Dossier 4D**). A noter que vous devez copier ces fichiers manuellement sur le poste distant.

Note : La présence du composant *4DSLI.DLL* est également nécessaire pour l'utilisation des commandes de cryptage des données **CRYPTER BLOB** et **DECRYPTER BLOB**.

L'installation de ces éléments rend possible l'utilisation du TLS dans les connexions au serveur Web 4D. Toutefois, pour que les connexions sécurisées soient acceptées par le serveur Web 4D, vous devez "activer" TLS. Ce paramètre est accessible dans la page **Web**, onglet **Configuration** des Propriétés de la base :



Par défaut, les connexions TLS sont activées. Vous pouvez désélectionner cette option si vous ne souhaitez pas exploiter les fonctionnalités TLS avec votre serveur Web, ou si un autre serveur autorisant les connexions sécurisées fonctionne sur le même poste.

Le port TCP utilisé pour les connexions TLS est par défaut le 443. Ce numéro de port peut être modifié dans la zone **Numéro de port HTTPS** afin, par exemple, de renforcer la sécurité du serveur Web (pour plus d'informations sur ce point, reportez-vous à la section **Paramétrages du serveur Web**). Le port TCP défini dans cette page de propriétés est utilisé pour les connexions du serveur Web en mode standard.

Note : Les autres propriétés de gestion du serveur Web 4D (mots de passe, délai avant déconnexion, taille du cache, etc.) restent appliquées, que le serveur fonctionne en mode TLS ou non.

Connexion des navigateurs en TLS

Pour qu'une connexion Web s'effectue en mode sécurisé, il suffit simplement que l'URL envoyé par le navigateur débute par "https" (au lieu de "http"). Dans ce cas, une boîte de dialogue d'alerte apparaît sur le navigateur. Une fois la boîte de dialogue validée, le certificat est envoyé au navigateur par le serveur Web.

Les deux parties "négocient" alors l'algorithme de cryptage qui va être utilisé pour la connexion.
Le serveur Web 4D dispose de plusieurs algorithmes de cryptage symétriques (RC2, RC4, DES...). L'algorithme commun le plus puissant est utilisé.

Attention : Le niveau de cryptage autorisé est soumis à la législation en vigueur dans le pays d'utilisation.

Contrôle du mode de connexion

L'utilisation de TLS dans le serveur Web 4D ne nécessite pas de configuration système particulière. Toutefois, vous devez tenir compte du fait qu'un serveur Web TLS peut également fonctionner en mode non sécurisé. Le changement de mode de connexion peut s'effectuer si le navigateur en fait la demande (il suffit à l'utilisateur, dans la zone d'URL du navigateur, de remplacer "HTTPS" par "HTTP"). Il revient au développeur d'interdire ou de rediriger les requêtes effectuées en mode non sécurisé — la routine **WEB Connexion securisee** permet de connaître le mode de connexion courant.

WML

Le serveur Web 4D supporte la technologie WML (Wireless Markup Language). Cette fonctionnalité autorise la consultation et la saisie d'informations dans des bases 4D à l'aide d'un téléphone mobile ou d'un assistant électronique personnel (PDA).

Note : Le langage WML, associé au protocole WAP (Wireless Application Protocol), est développé conjointement par plusieurs sociétés. Le WAP propose un ensemble d'outils de communication réseau, destiné à permettre aux téléphones mobiles et aux PDA de visualiser du texte publié dans des pages Web. La technologie WML est ouverte et libre de droits. Pour plus d'informations sur les spécifications du WML, veuillez consulter le site Web de Phone.com : *http://www.phone.com/*

La visualisation et la saisie d'informations s'effectuent par l'intermédiaire de pages WML utilisant le mécanisme des balises type *4DTEXT* ou *4DSCRIPT*.

Voici la liste des documents WML acceptés par le serveur Web 4D :

Extension	Type MIME	Description
.wml	text/vnd.wap.wml	Pages WML (toujours analysées par 4D*)
.wmls	text/vnd.wap.wmlscript	Scripts WML (côté client)
.wmlc	application/vnd.wap.wmlc	Pages WML binaires
.wmlsc	application/vnd.wap.wmlscript	Scripts WML binaires
.wbmp	image/vnd.wap.wbmp	Images bitmap destinées aux mobiles (parfois non prises en charge)

* Permet l'insertion dynamique de données via les balises type *4DTEXT* et *4DSCRIPT*.

XML

Le serveur Web 4D accepte les documents .xml, .xsl et .dtd. Ces documents sont respectivement envoyés avec le type MIME "text/xml", "text/xsl" et "text/xml". 4D analyse leur contenu et traite les éventuelles balises de type *4DTEXT* ou *4DSCRIPT* qu'ils contiennent, afin de générer du XML dynamique.

WEB ARRETER SERVEUR

WEB ARRETER SERVEUR
Ne requiert pas de paramètre

Description

La commande **WEB ARRETER SERVEUR** stoppe le serveur Web de l'application 4D sur laquelle elle a été exécutée (4D ou 4D Server).
Si le serveur Web était lancé, toutes les connexions Web sont interrompues et tous les process Web sont arrêtés.
Si le serveur Web n'était pas lancé, la commande ne fait rien.

WEB Connexion securisee

WEB Connexion securisee -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai = la connexion Web est sécurisée, Faux = la connexion Web n'est pas sécurisée

Description

La commande **WEB Connexion securisee** retourne un booléen indiquant si la connexion au serveur Web 4D s'effectue en mode sécurisé via TLS/SSL (la requête débute par "https:" au lieu de "http:").

- Si la connexion est effectuée en TLS ou SSL, la fonction retourne Vrai.
- Si la connexion est effectuée en mode classique (non sécurisé), la fonction retourne Faux.

Cette commande permet par exemple, le cas échéant, de refuser les tentatives de connexion en mode non sécurisé. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser le protocole TLS**.

WEB DEMARRER SERVEUR

WEB DEMARRER SERVEUR
Ne requiert pas de paramètre

Description

La commande **WEB DEMARRER SERVEUR** démarre le serveur Web de l'application 4D sur laquelle elle a été exécutée (4D ou 4D Server). La base est alors publiée sur votre réseau Intranet ou sur Internet.

Si le serveur Web a été correctement lancé, la variable système **OK** prend la valeur 1, sinon — si par exemple le protocole réseau TCP/IP n'est pas correctement configuré — **OK** prend la valeur 0 (zéro).

Variables et ensembles système

Si le serveur Web est correctement démarré, **OK** prend la valeur 1, sinon **OK** prend la valeur 0 (zéro).

WEB ENVOYER BLOB

WEB ENVOYER BLOB (blob ; type)

Paramètre	Type		Description
blob	BLOB	⇒	BLOB à envoyer au browser
type	Chaîne	⇒	Type de données du BLOB

Description

La commande **WEB ENVOYER BLOB** permet d'envoyer le BLOB *blob* au navigateur.

Le type de données contenues dans le BLOB est indiqué par le paramètre *type*. Ce paramètre peut contenir les valeurs suivantes :

- *type* = **Chaîne vide** ("") : dans ce cas, vous ne fournissez aucune information sur le BLOB. Le navigateur tentera alors d'interpréter lui-même le contenu du BLOB.
- *type* = **Extension de fichier** (ex. : ".HTM", ".GIF", ".JPEG", etc.) : dans ce cas, vous fournissez au navigateur, par l'intermédiaire de son extension, le type MIME des données contenues dans le BLOB. Le BLOB sera interprété en fonction de cette extension. Toutefois, l'extension doit être standard afin que le navigateur puisse l'interpréter correctement. Une liste des types MIME les plus courants et de leurs extensions est fournie ci-dessous.
- *type* = **Mime/Type** (ex. : "text/html", "image/tiff", etc.) : dans ce cas, vous fournissez directement au navigateur le type MIME des données contenues dans le BLOB. Cette solution est celle qui vous offre le plus de latitude. En effet, outre les types standard, vous pouvez passer un type MIME personnalisé pour envoyer des documents propriétaires en Intranet. Il vous suffit pour cela de configurer les navigateurs afin qu'ils reconnaissent le type envoyé et, par exemple, exécutent l'application correspondante. La valeur à passer dans le paramètre *type* est, dans ce cas "application/x-[NomDuType]"). Dans les navigateurs des postes clients, vous référencez ce type et lui associez l'action "Exécuter l'application". La commande **WEB ENVOYER BLOB** vous permet alors d'envoyer des documents de tout type, les clients Intranet ouvrant automatiquement l'application associée.

  Note : Pour plus d'informations sur les types MIME, reportez-vous à la page http://www.iana.org/assignments/media-types.

Voici une liste des types MIME les plus courants :

Extension	Mime/Type
.htm	text/html
.html	text/html
.shtml	text/html
.shtm	text/html
.css	text/css
.pdf	application/pdf
.rtf	application/rtf
.ps	application/postscript
.eps	application/postscript
.hqx	application/mac-binhex40
.js	application/javascript
.json	application/json
.txt	text/plain
.text	text/plain
.gif	image/gif
.jpg	image/jpeg
.jpeg	image/jpeg
.jpe	image/jpeg
.jfif	image/jpeg
.pic	image/pict
.pict	image/pict
.tif	image/tiff
.tiff	image/tiff
.mpeg	video/mpeg
.mpg	video/mpeg
.mov	video/quicktime
.moov	video/quicktime
.aif	audio/aiff
.aiff	audio/aiff
.wav	audio/wav
.ram	audio/x-pn-realaudio
.sit	application/x-stuffit
.bin	application/x-stuffit
.xml	application/xml
.z	application/x-zip
.zip	application/x-zip
.gz	application/x-gzip
.tar	application/x-tar

Note : La liste des types MIME pris en charge par le serveur HTTP de 4D est stockée dans le fichier "MimeTypes.xml" situé dans le dossier suivant de l'application 4D : *[Contents]\Native components\HTTPServer.bundle\Contents\Resources*.

Les éventuelles références aux variables 4D et balises de type *4DSCRIPT* dans la page sont toujours analysées.

Exemple

Reportez-vous à l'exemple de la routine **IMAGE VERS GIF**.

WEB ENVOYER DONNEES

WEB ENVOYER DONNEES (données {; *})

Paramètre	Type		Description
données	BLOB	⇒	Données HTTP à envoyer
*	Opérateur	⇒	Envoi morcelé (chunked)

Description

La commande **WEB ENVOYER DONNEES** permet au serveur Web 4D d'envoyer des données HTTP "brutes", pouvant être morcelées.

Le paramètre *données* contient les deux parties standard d'une réponse HTTP, c'est-à-dire l'en-tête et le corps (header et body). Les données sont envoyées sans formatage préalable par le serveur. Toutefois, 4D effectue un contrôle élémentaire sur l'en-tête et le corps de la réponse afin qu'elle soit valide :

- Si l'en-tête est incomplet ou non conforme aux spécifications du protocole HTTP, 4D le modifie en conséquence.
- Si la réponse HTTP est incomplète, 4D ajoute les informations manquantes. Si, par exemple, vous souhaitez effectuer une redirection, vous devez écrire :

```
HTTP/1.1 302 Location : http://...
```

Si vous passez uniquement :

```
Location : http://...
```

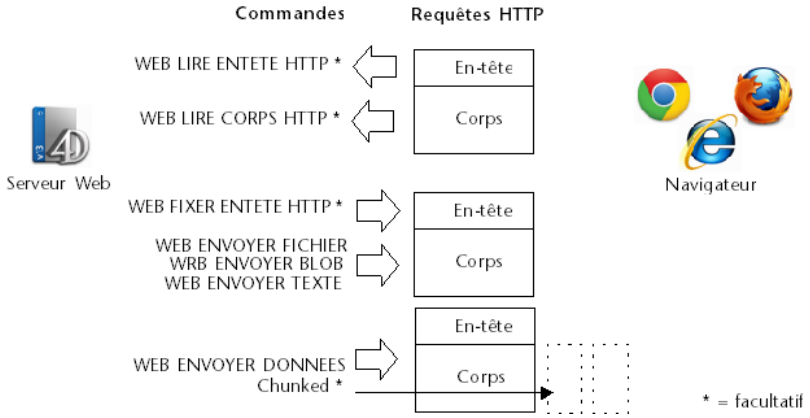
4D complétera la réponse en ajoutant **HTTP/1.1 302**.

Le paramètre optionnel * permet de déclarer que la réponse sera envoyée sous forme "morcelée" (chunked). Le découpage des réponses peut être utile lorsque le serveur envoie une réponse sans connaître sa longueur totale (par exemple si la réponse n'a pas encore été générée). Tous les navigateurs compatibles HTTP/1.1 acceptent les réponses "morcelées".
Si vous passez le paramètre *, le serveur Web inclura automatiquement le champ *transfer-encoding: chunked* dans l'en-tête de la réponse, si nécessaire (vous pouvez gérer manuellement l'en-tête de la réponse si vous le souhaitez). Le reste de la réponse sera également formaté en respectant la syntaxe de l'option chunked. Les réponses morcelées comportent un seul en-tête et un nombre indéfini de corps.
Toutes les instructions **WEB ENVOYER DONNEES** suivant l'exécution de **WEB ENVOYER DONNEES**(données;*) au sein de la même méthode seront considérées comme partie de la réponse (qu'elles contiennent ou non le paramètre *). Le serveur met un terme à l'envoi morcelé à la fin de l'exécution de la méthode.

Note : Si le client Web ne prend pas en charge le protocole HTTP/1.1, 4D convertira automatiquement la réponse au format compatible HTTP/1.0 (l'envoi ne sera pas morcelé). Dans ce cas toutefois, il est possible que le résultat ne corresponde pas à vos attentes. Il est donc recommandé de tester si le navigateur est compatible HTTP/1.1 et d'envoyer une réponse adaptée. Pour cela, vous pouvez utiliser une méthode de ce type :

```
C_BOOLEEN($0)
TABLEAU TEXTE(tabChamps;0)
TABLEAU TEXTE(tabValeurs;0)
WEB LIRE ENTETE HTTP(tabChamps;tabValeurs)
$0:=Faux
Si(Taille tableau(tabValeurs)>=3)
   Si(Position("HTTP/1.1";tabValeurs{3})>0)
      $0:=Vrai //Le navigateur est compatible HTTP/1.1, on retourne Vrai dans $0
   Fin de si
Fin de si
```

Combinée à la commande **WEB LIRE CORPS HTTP** et aux autres commandes du thème "Serveur Web", cette commande complète la gamme d'outils mis à la disposition des développeurs 4D pour traiter de manière entièrement personnalisée les connexions HTTP entrantes et sortantes. Ces différents outils sont présentés dans le schéma suivant :



Exemple

Cet exemple illustre l'emploi de l'option chunked avec la commande **WEB ENVOYER DONNEES**. Les données (une suite de chiffres) sont envoyées en

100 morceaux générés à la volée dans une boucle. A noter que l'en-tête de la réponse n'est pas explicitement défini : la commande **WEB ENVOYER DONNEES** l'enverra automatiquement et y insérera le champ *transfer-encoding: chunked* car le paramètre * est utilisé.

```
C_ENTIER LONG($cpt)
C_BLOB($mon_blob)
C_TEXTE($output)

Boucle($cpt;1;100)
   $output:="["+Chaine($cpt)+"]"
   TEXTE VERS BLOB($output;$mon_blob;UTF8 texte sans longueur)
   WEB ENVOYER DONNEES($mon_blob;*)
Fin de boucle
```

WEB ENVOYER FICHIER

WEB ENVOYER FICHIER (fichierWeb)

Paramètre	Type		Description
fichierWeb	Chaîne	⇨	Chemin d'accès au fichier Web à envoyer

Description

La commande **WEB ENVOYER FICHIER** envoie au navigateur Web la page HTML ou le fichier Web stocké dans le document dont vous passez le chemin d'accès dans *fichierHTML*. La commande peut envoyer tout type de fichier pris en charge par les navigateurs Web (pages html mais aussi fichiers xml ou txt, images jpeg, tiff...)

Par défaut, 4D recherche le document à l'intérieur du dossier racine HTML, défini dans les Propriétés de la base.
Cette commande accepte en paramètre un chemin d'accès exprimé en syntaxe Posix (noms de répertoires ou de dossiers séparés par une barre oblique "/") ou en syntaxe système.
Si vous passez un chemin d'accès invalide, une erreur liée à la gestion de fichiers de votre système d'exploitation est générée. Vous pouvez intercepter l'erreur à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**. Si la méthode affiche une boîte de dialogue d'alerte ou de message, celle-ci apparaît sur le poste du navigateur.

Une fois que l'instruction **WEB ENVOYER FICHIER** a été exécutée, la variable système OK est mise à jour : si le fichier à envoyer existe et si le timeout n'est pas dépassé, la variable OK prend la valeur 1. Dans le cas contraire, elle prend la valeur 0.

Note : Si vous appelez **WEB ENVOYER FICHIER** depuis un process qui n'est pas un process Web, la commande ne fait rien. Aucune erreur n'est retournée, l'appel est simplement ignoré.

Les éventuelles références aux variables 4D et aux balises de type *4DSCRIPT* présentes dans la page sont analysées lorsque le type du document le permet (document basé sur du texte).

Exemple

Le dossier racine HTML de la base est le dossier **WebDocs**. Il contient les éléments suivants :

```
..\WebDocs\HTM\MaPage.HTM
```

L'envoi de la page Web *"MaPage.HTM"* doit être effectué de cette manière :

```
WEB ENVOYER FICHIER("HTM/MaPage.HTM")
```

Variables et ensembles système

Si le fichier à envoyer existe et si le timeout n'est pas dépassé, la variable OK prend la valeur 1. Dans le cas contraire, elle prend la valeur 0.

WEB ENVOYER REDIRECTION HTTP

WEB ENVOYER REDIRECTION HTTP (url {; *})

Paramètre	Type		Description
url	Chaîne	⇒	Nouvel URL
*	Opérateur	⇒	Si spécifié = l'URL n'est pas traduit, Si omis = l'URL est traduit

Description

La commande **WEB ENVOYER REDIRECTION HTTP** permet de transformer un URL en un autre.

Le paramètre *url* contient le nouvel URL qui permet de rediriger la requête. Si ce paramètre est un url vers un fichier, il doit contenir la référence à ce fichier, par exemple : **WEB ENVOYER REDIRECTION HTTP** ("/MaPage.HTM").

Cette commande prévaut sur les commandes d'envoi de données (**WEB ENVOYER FICHIER**, **WEB ENVOYER BLOB**, etc.) éventuellement placées dans la même méthode.

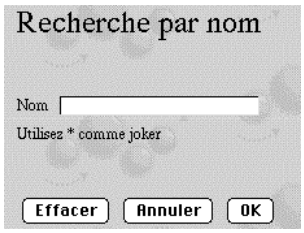
Cette commande permet également de rediriger une requête vers un autre serveur Web.

4D encode automatiquement les caractères spéciaux de l'URL. Si vous passez le caractère *, 4D ne traduira pas les caractères spéciaux de l'URL.

A noter que le statut de la requête envoyée par cette commande est **302 : redirection temporaire**. Si vous avez besoin d'une redirection permanente (statut 301), vous pouvez fixer le champ HTTP *X-STATUS: 301* dans l'en-tête de la réponse.

Exemple

Vous pouvez utiliser cet URL pour effectuer, à l'aide de pages statiques, des recherches personnalisées dans 4D. Imaginez que vous ayez placé dans une page HTML statique les éléments suivants :



L'action POST "/4dcgi/rech" a été associée à la zone de texte et aux boutons **OK** et **Annuler**.
Dans la **Méthode base Sur connexion Web**, placez les instructions suivantes :

```
Au cas ou
   :($1="/4dcgi/rech") `Lorsque 4D reçoit cet URL
  `Si le bouton OK a été utilisé et le champ 'nom' contient une valeur
      Si((bOK="OK") & (nom # ""))
  `Transformer l'URL afin d'exécuter le code de la recherche, placé plus
  `loin dans la même méthode
         WEB ENVOYER REDIRECTION HTTP("/4dcgi/rech?"+nom)
      Sinon `Sinon retourner à la page de départ
         WEB ENVOYER REDIRECTION HTTP("/page1.htm")
      Fin de si
      ...

   :($1="/4dcgi/rech?@") `Si l'URL a été redirigé
      ... `Placez ici le code de la recherche
Fin de cas
```

WEB ENVOYER TEXTE

WEB ENVOYER TEXTE (texteHTML {; type})

Paramètre	Type		Description
texteHTML	Texte	➡	Champ ou variable texte au format HTML à envoyer au navigateur
type	Texte	➡	Type MIME

Description

La commande **WEB ENVOYER TEXTE** permet d'envoyer directement des données texte formatées en HTML.

Le paramètre *texteHTML* contient les données à envoyer. 4D n'effectue aucun contrôle sur le contenu de ce paramètre, vous devez donc veiller à ce que le codage HTML soit correct.

Les éventuelles références aux variables 4D et balises de type *4DSCRIPT* dans le texte sont toujours analysées.

Par défaut, si vous omettez le paramètre *type*, 4D assume que les données envoyées sont du type "text/html". La commande équivaut alors strictement à l'envoi d'un BLOB ayant le type "text/html" à l'aide de la commande **WEB ENVOYER BLOB**.

Vous pouvez également préciser dans *type* le type MIME du texte à envoyer. Pour plus d'informations sur les types MIME pris en charge, reportez-vous à la description de la commande **WEB ENVOYER BLOB**.

Exemple

La méthode suivante :

```
TEXTE VERS BLOB("<html><head></head><body>"+Chaine(Heure courante)+"</body></html>";$blob;UTF8 texte sans longueur)
WEB ENVOYER BLOB($blob;"text/html")
```

... peut être remplacée par :

```
WEB ENVOYER TEXTE("<html><head></head><body>"+Chaine(Heure courante)+"</body></html>")
```

WEB FERMER SESSION

WEB FERMER SESSION (idSession)

Paramètre	Type		Description
idSession	Texte	⇨	UUID de session

Description

La commande **WEB FERMER SESSION** clôt la session Web existante désignée par le paramètre *idSession*. Si la session n'existe pas, la commande ne fait rien.

Lorsque cette commande est appelée depuis un process Web ou tout autre process :

- la date d'expiration du cookie est mise à 0,
- la **Méthode base Sur fermeture process Web** est appelée, vous permettant de stocker les informations de la session,
- les sélections courantes sont détruites, les enregistrements déverrouillés et les variables réinitialisées.

Après l'exécution de cette commande, si un client Web envoie une requête utilisant un cookie invalide, une nouvelle session est ouverte et un nouveau cookie est envoyé.

Note : Dans le contexte d'une session 4D Mobile, la commande **WEB FERMER SESSION** referme la session 4D Mobile dont l'ID a été passé dans *idSession*. Comme une session 4D Mobile peut gérer plusieurs process, cette commande demande en fait à tous les process Web liés à la session de terminer leur exécution.

WEB FIXER ENTETE HTTP

WEB FIXER ENTETE HTTP (entête | tabChamps {; tabValeurs})

Paramètre	Type		Description
entête | tabChamps	Texte, Tableau texte	⇒	Champ ou variable contenant l'en-tête HTTP de la requête ou Tableau des champs de l'en-tête HTTP
tabValeurs	Tableau texte	⇒	Contenu des champs de l'en-tête HTTP

Description

La commande **WEB FIXER ENTETE HTTP** permet de fixer les champs de l'en-tête HTTP de la réponse faite au navigateur Web par 4D. Elle n'a d'effet que dans un process Web.
Cette commande vous permet, en particulier, de gérer des "cookies".
WEB FIXER ENTETE HTTP admet deux syntaxes :

- **Première syntaxe : WEB FIXER ENTETE HTTP (entête)**

Vous passez dans le paramètre *entête*, de type variable ou champ texte, les champs de l'en-tête HTTP que vous souhaitez fixer. Les champs doivent être séparés entre eux par un retour chariot ou une séquence cr/lf (retour chariot/retour à la ligne), sous Windows et Mac OS, 4D se charge du formatage de la réponse.
Voici un exemple de "cookie" personnalisé :

```
C_TEXTE($vTcookie)
$vTcookie:="Set-Cookie: USER="+Chaine(Abs(Hasard))+"; PATH=/"
WEB FIXER ENTETE HTTP($vTcookie)
```

Note : Il n'est pas possible de passer une constante texte littérale directement dans le paramètre *entête*. Vous devez utiliser une variable ou un champ intermédiaire.

Pour plus d'informations sur la syntaxe à appliquer dans les en-têtes HTTP, veuillez consulter sur Internet les R.F.C (Request For Comments) à l'adresse http://www.w3c.org.

- **Deuxième syntaxe : WEB FIXER ENTETE HTTP (tabChamps; tabValeurs)**

L'en-tête HTTP est défini à l'aide de deux tableaux texte, *tabChamps* et *tabValeurs*.
L'en-tête sera écrit de la manière suivante :

```
tabChamps{1}:="X-VERSION"
tabChamps{2}:="X-STATUS"
tabChamps{3}:="Set-Cookie"
tabChamps{4}:="Server"

tabValeurs{1}:="HTTP/1.0"*
tabValeurs{2}:="200 OK"*
tabValeurs{3}:="C=HELLO"
tabValeurs{4}:="North_Carolina"
```

* Ces deux premiers éléments constituent la première ligne de la réponse. Lorsqu'ils sont saisis, ils doivent impérativement être les éléments 1 et 2 des tableaux. Il est toutefois possible de les omettre et d'écrire seulement — 4D se chargeant de formater l'en-tête :

```
tabChamps{1}:="Set-Cookie"
tabValeurs{1}:="C=HELLO"
```

Si vous ne spécifiez pas de statut, celui-ci est automatiquement HTTP/1.0 200 OK. Le champ **Server** est par défaut "4D/<version>". Les champs **Content-Length** et **Date** sont également définis par défaut par 4D.

WEB FIXER OPTION

WEB FIXER OPTION (sélecteur ; valeur)

Paramètre	Type		Description
sélecteur	Entier long	➡	Code de l'option à modifier
valeur	Entier long, Texte	➡	Valeur de l'option

Description

La commande **WEB FIXER OPTION** permet de modifier la valeur courante de diverses options de fonctionnement du serveur Web de 4D.

Passez dans le paramètre *sélecteur* une des constantes du thème "**Serveur Web**" et dans *valeur* la nouvelle valeur de l'option :

Constante	Type	Valeur	Comment
Web activer validation adresse IP de session	Entier long	83	**Portée** : serveur Web local **Conservé entre deux sessions** : Non **Description** : Permet de désactiver ou d'activer la validation des adresses IP pour les cookies de session. Pour des raisons de sécurité, par défaut le serveur Web de 4D vérifie l'adresse IP de chaque requête contenant un cookie de session et rejette la requête si cette adresse ne correspond pas à l'adresse IP utilisée pour créer le cookie. Dans certaines applications spécifiques, vous pouvez souhaiter désactiver cette validation et accepter les cookies de session même si les adresses IP ne correspondent pas. Par exemple, lorsque les appareils mobiles basculent du réseau Wifi au réseau 3G/4G, leur adresse IP change. Dans ce cas, vous pouvez passer 0 dans cette option afin de permettre aux clients de pouvoir continuer à utiliser leur session Web même après changement d'adresse IP. Notez que ce paramétrage abaisse le niveau de sécurité de votre application. Lorsqu'il est modifié, ce paramétrage est effectif immédiatement (il n'est pas nécessaire de redémarrer le serveur HTTP). **Valeurs possibles** : 0 (désactivé) ou 1 (activé) **Valeur par défaut** : 1 (les adresses IP sont vérifiées)
Web adresse IP d'écoute	Entier long	16	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Adresse IP sur laquelle le serveur Web doit recevoir les requêtes HTTP avec 4D en mode local et 4D Server. Par défaut, aucune adresse particulière n'est spécifiée (*valeur* = 0). Ce paramètre est défini dans les Propriétés de la base. Le sélecteur *Web Adresse IP d'écoute* est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d'accès au mode Développement). Le paramètre *valeur* contient l'adresse IP sous forme hexadécimale. Ainsi, pour désigner une adresse du type "a.b.c.d", le code sera de la forme : `C_ENTIER LONG($addr)` `$addr:=($a<<24)\|($b<<16)|($c<<8)\|$d` `WEB FIXER OPTION(Web Adresse IP découte;$addr)`
Web chemin du cookie de session	Entier long	82	**Portée :** serveur Web local **Conservé entre deux sessions :** Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de fixer ou de lire la valeur du champ "path" du cookie de session. Ce sélecteur (ainsi que le sélecteur 81) est utile pour contrôler la portée des cookies de session : si vous définissez par exemple la valeur "/4DACTION" pour ce sélecteur, le client n'enverra un cookie que pour les requêtes dynamiques débutant par 4DACTION et pas pour les images, pages statiques, etc. **Valeurs :** Texte
Web conserver les sessions	Entier long	70	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet d'activer ou d'inactiver la gestion automatique des sessions (décrit dans la section **Gestion des sessions Web**). **Valeurs** : 1 (activer mode) ou 0 (inactiver mode) **Valeur par défaut** : 1 pour les bases créées depuis la v13, 0 pour les bases converties. A noter que ce mode active également le mécanisme de réutilisation des contextes temporaires en mode distant. Pour plus d'informations sur ce mécanisme, reportez-vous à la description de cette option dans la section **Paramétrages du serveur Web**.
Web debug log	Entier long	84	**Portée :** serveur Web local **Conservé entre deux sessions :** Non, mais reste valide même si le serveur HTTP est redémarré (un nouveau fichier est utilisé) **Description :** Permet de lire ou de définir le statut du fichier de debug de requêtes HTTP du serveur Web 4D. Lorsqu'il est activé, ce fichier, nommé "**HTTPDebugLog_*nn*.txt**", est stocké dans le dossier "Logs" de l'application (*nn* est le numéro de fichier). L'historique des requêtes HTTP est particulièrement utile dans le contexte du débogage du serveur Web. Il stocke en texte brut chaque requête et chaque réponse. La totalité des requêtes, en-têtes compris, est enregistrée ; en option, il est possible d'enregistrer également le corps (body) des requêtes. **Valeurs** : Une des constantes préfixées "wdl" (reportez-vous ci-dessous à la description de ces constantes). **Valeur par défaut** : 0 (non activé)
Web domaine du cookie de session	Entier long	81	**Portée :** serveur Web local **Conservé entre deux sessions :** Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de fixer ou de lire la valeur du champ "domain" du cookie de session. Ce sélecteur (ainsi que le sélecteur 82) est utile pour contrôler la portée des cookies de session : si vous définissez par exemple la valeur "/*.4d.fr" pour ce sélecteur, le client n'enverra un cookie que lorsque la requête s'adresse au domaine ".4d.fr", ce qui exclut les serveurs hébergeant des données statiques externes. **Valeurs :** Texte
			Portée : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Démarrage ou arrêt de l'enregistrement des requêtes Web reçues par le serveur Web de 4D en mode local ou de 4D Server. Par défaut, la valeur est 0 (pas d'enregistrement des requêtes).

Constante	Type	Valeur	Commentaire
Web enreg requêtes	Entier long	29	L'historique des requêtes Web est stocké sous la forme d'un fichier texte nommé "logweb.txt" automatiquement placé dans le dossier Logs de la base, à côté du fichier de structure. Le format de ce fichier est déterminé par la valeur que vous passez. Pour plus d'informations sur les différents formats d'historique des requêtes, reportez-vous à la section **Informations sur le site Web**. L'activation de ce fichier peut également être définie dans la page "Web/Journal" des Propriétés de la base. **Valeurs possibles** : 0 = Ne pas enregistrer (défaut), 1 = Enregistrer au format CLF, 2 = Enregistrer au format DLF, 3 = Enregistrer au format ELF, 4 = Enregistrer au format WLF. **Attention** : Les formats 3 et 4 sont des formats personnalisés, dont le contenu doit être défini au préalable dans les Propriétés de la base. Si vous utilisez l'un de ces formats sans que ses champs n'aient été sélectionnés, le fichier des requêtes n'est pas généré. **Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui
Web jeu de caractères	Entier long	17	**Description** : Jeu de caractères que le serveur Web 4D (avec 4D en mode local et 4D Server) utilise pour communiquer avec les navigateurs Web qui se connectent à la base. La valeur par défaut dépend de la langue du système d'exploitation. Ce paramètre est défini dans les Propriétés de la base. **Valeurs** : Les valeurs possibles dépendent du mode d'exécution de la base relatif au jeu de caractères. • *Mode Unicode* : Lorsque l'application est exécutée en mode Unicode, les valeurs à passer pour ce paramètre sont des identifiants de jeux de caractères (entier long *MIBEnum* ou chaîne *Name*, identifiants définis par l'IANA, cf. http://www.iana.org/assignments/character-sets). Voici la liste des identifiants correspondant aux jeux de caractères pris en charge par le serveur Web de 4D : 4=ISO-8859-1 12=ISO-8859-9 13=ISO-8859-10 17=Shift-JIS 2024=Windows-31J 2026=Big5 38=euc-kr 106=UTF-8 2250=Windows-1250 2251=Windows-1251 2253=Windows-1253 2255=Windows-1255 2256=Windows-1256 • *Mode compatibilité ASCII* : 0 : Occidental 1 : Japonais 2 : Chinois 3 : Coréen 4 : Défini par l'utilisateur 5 : Réservé 6 : Europe Centrale 7 : Cyrillique 8 : Arabe 9 : Grec 10 : Hébreu 11 : Turc 12 : Nordique
Web niveau de compression HTTP	Entier long	50	**Portée** : Serveur Web local **Conservé entre deux sessions** : Non **Description** : Niveau de compression pour tous les échanges HTTP compressés effectués pour le serveur HTTP de 4D (requêtes client ou réponses serveur, Web et Web Service). Ce sélecteur permet d'optimiser les échanges en privilégiant la rapidité d'exécution (compression moindre) ou la quantité de compression (vitesse moindre). Le choix d'une valeur dépend de la taille et de la nature des données échangées. Passez de 1 à 9 dans le paramètre *valeur*, 1 étant la compression la plus rapide et 9 la plus élevée. Vous pouvez également passer -1 pour obtenir un compromis entre vitesse et taux de compression. Par défaut le niveau de compression est 1 (compression rapide). **Valeurs** : 1 à 9 (1 = plus rapide, 9 = plus compressé) ou -1 = meilleur compromis.
Web nom du cookie de session	Entier long	73	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de définir le nom du cookie utilisé pour stocker l'ID de session. **Valeurs :** Texte. **Valeur par défaut :** "4DSID" (passez une chaîne vide pour rétablir la valeur par défaut)
Web nombre de sessions max	Entier long	71	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description** : Permet de limiter le nombre de sessions simultanées. Lorsque le nombre défini est atteint, la session la plus ancienne est détruite (et la **Méthode base Sur fermeture process Web** est appelée) si le serveur Web a besoin d'en créer une nouvelle. **Valeurs** : Entier long. Le nombre de sessions simultanées ne peut pas dépasser le nombre total de process Web (option Web process Web simultanés maxi, 100 par défaut) **Valeur par défaut** : 100 (passez 0 pour rétablir la valeur par défaut)
Web numéro de port HTTPS	Entier long	39	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Numéro du port TCP utilisé par le serveur Web de 4D en mode local et de 4D Server pour les connexions sécurisées via SSL (protocole HTTPS). Le numéro de port HTTPS est défini dans la page "Web/Configuration" de la boîte de dialogue des Propriétés de la base. Par défaut, la valeur est 443 (valeur standard). Vous pouvez utiliser les constantes du thème **Numéros de port TCP** pour le paramètre *valeur*. **Valeurs possibles** : 0 à 65535 Portée : 4D en mode local et 4D Server.

Constante	Type	Valeur	Comment
Web numéro du port	Entier long	15	**Conservé entre deux sessions :** Non **Description :** Permet de fixer ou de lire le numéro du port TCP utilisé par le serveur Web 4D avec 4D en mode local et 4D Server. Le numéro de port TCP est défini dans la page “Web/Configuration” de la boîte de dialogue des Propriétés de la base. Vous pouvez utiliser les constantes du thème **Numéros de port TCP** pour le paramètre *valeur*. Ce sélecteur est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d’accès au mode Développement). **Valeurs :** Pour plus d’informations sur le numéro de port TCP, reportez-vous à la section **Paramétrages du serveur Web**. **Valeur par défaut :** 80
Web process Web simultanés maxi	Entier long	18	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Limite strictement supérieure du nombre de process Web de tout type acceptés par le serveur Web avec 4D en mode local et 4D Server. Lorsque ce nombre limite (moins un) est atteint, 4D ne crée plus de nouveau process et retourne le message “Serveur non disponible” (statut HTTP 503 - Service Unavailable) à toute nouvelle requête. Ce paramètre permet de prévenir la saturation du serveur Web 4D pouvant se produire lors d’un envoi massif de requêtes ou d’une demande excessive de création de contextes. Il peut également être défini dans la boîte de dialogue des Propriétés de la base. En théorie, le nombre maximum de process Web est le résultat de la division Mémoire disponible / Taille de la pile d’un process Web. Une autre solution consiste à visualiser les informations sur les process Web affichées dans l’Exporateur d’exécution : le nombre courant de process Web et le nombre maximum atteint depuis le démarrage du serveur Web sont indiqués. **Valeurs** : Toute valeur incluse entre 10 et 32 000. La valeur par défaut est 100.
Web seuil de compression HTTP	Entier long	51	**Portée** : Serveur HTTP local **Conservé entre deux sessions** : Non **Description** : Dans le cadre d’échanges HTTP optimisés, seuil de taille de requête au-dessous de laquelle les échanges ne doivent pas être compressés. Ce paramétrage est utile pour éviter de perdre du temps machine à compresser les trop petits échanges. **Valeurs possibles** : Toute valeur de type Entier long. Le paramètre *valeur* contient une taille exprimée en octets. Par défaut, le seul de compression est fixé à 1024 octets.
Web taille max requêtes	Entier long	27	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Taille maximale (en octets) des requêtes HTTP entrantes (POST) que le serveur Web est autorisé à traiter. Par défaut, la valeur est 2 000 000, c’est-à-dire un peu moins de 2 Mo. La valeur maximale (2 147 483 648) signifie en pratique qu’aucune limite n’est fixée. Ce paramétrage permet d’empêcher la saturation du serveur Web à cause de requêtes entrantes de trop grande taille. Lorsqu’une requête atteint la limite, le serveur Web de 4D la refuse. **Valeurs possibles** : 500 000 à 2 147 483 648.
Web timeout process	Entier long	78	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de modifier la durée de vie des process inactifs associés aux sessions. A l’issue du timeout, le process est tué sur le serveur, la **Méthode base Sur fermeture process Web** est appelée puis le contexte de la session est détruit. **Valeurs** : Entier long (minutes) **Valeur par défaut** : 480 minutes (passez 0 pour rétablir la valeur par défaut)
Web timeout session	Entier long	72	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description** : Permet de modifier la durée de vie des sessions inactives (durée définie dans le cookie). A l’issue de cette durée, le cookie de session expire et n’est plus envoyé par le client HTTP. **Valeurs** : Entier long (minutes) **Valeur par défaut** : 480 minutes (passez 0 pour rétablir la valeur par défaut)
Web TRACE HTTP	Entier long	85	**Portée** : serveur Web local **Conservé entre deux sessions** : Non **Description** : Permet de désactiver ou d'activer la méthode HTTP TRACE dans le serveur Web de 4D. Pour des raisons de sécurité, à compter de 4D v15 R2, par défaut le serveur Web de 4D Web rejette les requêtes HTTP TRACE avec l'erreur 405 (see Désactivation de HTTP TRACE). Si nécessaire, vous pouvez rétablir la prise en charge de la méthode HTTP TRACE pour la session en passant cette constante avec la valeur 1. Lorsque l'option est activée, le serveur Web de 4D répond aux requêtes HTTP TRACE en retournant la ligne de requête, l'en-tête et le corps. **Valeurs possibles** : 0 (désactivé) ou 1 (activé) **Valeur par défaut** : 0 (désactivé)

Lorsque vous utilisez le *sélecteur* Web debug log, vous pouvez passer une des constantes suivantes dans le paramètre *valeur*:

Constante	Type	Valeur	Comment
wdl activer avec body request	Entier long	5	Le fichier de debug de requêtes HTTP Web est activé avec le body de la requête uniquement
wdl activer avec body response	Entier long	3	Le fichier de debug de requêtes HTTP Web est activé avec le body de la réponse uniquement
wdl activer avec tous body	Entier long	7	Le fichier de debug de requêtes HTTP Web est activé avec les body de la requête et de la réponse
wdl activer sans body	Entier long	1	Le fichier de debug de requêtes HTTP Web est activé sans les body (la taille des body est fournie dans ce cas)
wdl désactiver log	Entier long	0	Le fichier de debug de requêtes HTTP Web est désactivé

Exemple

Activation du fichier d'historique de debug des requêtes HTTP, sans les parties body :

```
WEB FIXER OPTION(Web debug log;wdl activer sans body)
```

Voici un exemple d'entrée enregistrée dans le fichier d'historique :

```
# REQUEST
# SocketID: 1592
# PeerIP: 127.0.0.1
# PeerPort: 54912
# TimeStamp: 39089388
GET /4DWEBTEST HTTP/1.1
Connection: Close
Host: 127.0.0.1
User-Agent: 4D_HTTP_Client/0.0.0.0

# RESPONSE
# SocketID: 1592
# PeerIP: 127.0.0.1
# PeerPort: 54912
# TimeStamp: 39089389 (elapsed time: 1 ms)
HTTP/1.1 200 OK
Accept-Ranges: bytes
Connection: close
Content-Length: 3555
Content-Type: text/plain; charset=UTF-8
Date: Tue, 20 Jan 2015 10:51:29 GMT
Expires: Tue, 20 Jan 2015 10:51:29 GMT
Pragma: no-cache
Server: 4D/14.6.0

[Body Size: 3555]
```

WEB FIXER PAGE ACCUEIL

WEB FIXER PAGE ACCUEIL (homePage)

Paramètre	Type		Description
homePage	Chaîne	⇒	Nom de page ou chemin d'accès HTML à la page ou "" pour ne pas envoyer de page d'accueil personnalisée

Description

La commande **WEB FIXER PAGE ACCUEIL** vous permet de modifier la page d'accueil (page Home) personnalisée pour le process Web courant.

La page définie est liée au process Web, vous pouvez donc définir des pages d'accueil différentes en fonction, par exemple, de l'utilisateur connecté. Cette page peut être statique ou semi-dynamique.

Vous passez dans le paramètre *homePage* le nom de la page HTML d'accueil ou le chemin d'accès HTML complet à la page.

Note : Si la page définie par le paramètre *homePage* n'existe pas lorsque le process Web y accède pour la première fois, le serveur Web la crée et lui affecte le contenu de la **Page d'accueil par défaut** .

Pour ne plus envoyer *homePage* comme page d'accueil pour le process Web courant, appelez de nouveau la commande **WEB FIXER PAGE ACCUEIL** en passant une chaîne vide ("") dans *homePage*.

Note : La page d'accueil par défaut du serveur Web est définie dans les Propriétés de la base.

WEB FIXER RACINE

WEB FIXER RACINE (dossierRacine)

Paramètre	Type		Description
dossierRacine	Chaîne	⇨	Chemin d'accès du dossier racine du serveur Web

Description

La commande **WEB FIXER RACINE** permet de modifier le dossier racine par défaut dans lequel 4D ira rechercher les fichiers HTML demandés au serveur Web.

Cette commande ne tient pas compte du dossier racine HTML par défaut éventuellement défini dans les Propriétés de la base. Pour plus d'informations sur ce dossier, reportez-vous à la section **Sécurité des connexions**.

L'emplacement du dossier racine peut être exprimé soit en syntaxe HTML (type URL), soit en syntaxe système (chemin absolu) :

- Syntaxe HTML : les noms de dossiers sont séparés par une barre oblique ("/"), quelle que soit la plate-forme que vous utilisez.
- Syntaxe système : chemin d'accès absolu ("nom long") respectant la syntaxe de la plate-forme courante, par exemple :
  - (Mac OS) Disque:Applications:monserv:dossier
  - (Windows) C:\Applications\monserv\dossier

Notes :

- La prise en compte du nouveau dossier racine nécessite le redémarrage du serveur Web.
- Vous pouvez connaître à tout moment l'emplacement du dossier racine courant à l'aide de la commande **Dossier 4D**.

Si vous passez un chemin d'accès invalide, une erreur liée à la gestion de fichiers du système d'exploitation est générée. Vous pouvez intercepter l'erreur à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**. Si la méthode affiche une boîte de dialogue d'alerte ou de message, celle-ci apparaît sur l'écran du navigateur.

WEB LIRE CORPS HTTP

WEB LIRE CORPS HTTP (corps)

Paramètre	Type		Description
corps	BLOB, Texte	⇐	Champ corps (Body) de la requête HTTP

Description

La commande **WEB LIRE CORPS HTTP** retourne le corps (body) de la requête HTTP en cours de traitement. Le corps HTTP est retourné tel quel, sans traitement ni analyse.

Cette commande peut être appelée depuis la **Méthode base Sur authentification Web**, la **Méthode base Sur connexion Web** ou toute méthode Web.

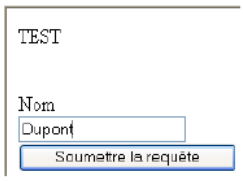
Vous pouvez passer dans le paramètre *corps* une variable ou un champ de type BLOB ou Texte. Le type Texte sera généralement suffisant (le paramètre *corps* peut recevoir jusqu'à 2 Go de texte).

Cette commande permet par exemple d'effectuer des recherches dans le corps des requêtes. Elle permet également aux utilisateurs avancés de mettre en place un serveur WebDAV au sein d'une base 4D.

Exemple

Dans cet exemple, une requête simple est envoyée au serveur Web de 4D et le contenu du champ HTTP corps est visualisé dans le débogueur. Voici le formulaire envoyé au serveur Web de 4D, ainsi que le code HTML correspondant :

Formulaire



Corps

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org
<html>
        <head>
        <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
        <title>Page de test</title>
        </head>

        <body>
        TEST
        <FORM ACTION="/4DACTION/Test4D2004" METHOD=POST>
        <br>Nom</br>
        <input type="text" value="saisissez votre nom" name="vnom">
        <br>
        <input type="submit">
        </FORM>
        </body>

</html>
```

Voici la méthode Test4D2004 :

```
C_BLOB($requete)
C_TEXTE($texteRequete)

WEB LIRE CORPS HTTP($requete)
$texteRequete:=BLOB vers texte($requete;UTF8 texte sans longueur)
WEB ENVOYER FICHIER("page.html")
```

Note : Cette méthode a été déclarée "Disponible via les balises HTML et les URLs 4D (4DACTION...)" dans ses propriétés.

Lorsque le formulaire est soumis au serveur Web, la variable $texteRequete reçoit le texte du champ body de la requête HTTP, soit "*vnom=Dupont*".

WEB LIRE ENTETE HTTP

WEB LIRE ENTETE HTTP (entête | tabChamps {; tabValeurs})

Paramètre	Type		Description
entête | tabChamps	Texte, Tableau texte	⇐	En-tête HTTP de la requête ou Champs de l'en-tête HTTP
tabValeurs	Tableau texte	⇐	Contenu des champs de l'en-tête HTTP

Description

La commande **WEB LIRE ENTETE HTTP** retourne, soit sous forme de chaîne, soit sous forme de deux tableaux, l'en-tête HTTP de la requête en cours de traitement.

Cette commande peut être appelée depuis toute méthode (**Méthode base Sur authentification Web**, **Méthode base Sur connexion Web**, méthode appelée par "*/4DACTION*"...) exécutée dans un process Web.

- **Première syntaxe : WEB LIRE ENTETE HTTP (entête)**

Lorsque vous utilisez cette syntaxe, le résultat retourné dans la variable *entête* est du type suivant :

```
"GET /page.html HTTP\1.0"+Caractere(13)+Caractere(10)+"User-Agent: browser"+Caractere(13)+Caractere(10)+"Cookie:
C=HELLO"
```

Chaque champ d'en-tête est séparé par une séquence CR+LF (Retour chariot+Retour à la ligne), sous Windows et Mac OS.

- **Seconde syntaxe : WEB LIRE ENTETE HTTP (tabChamps; tabValeurs)**

Lorsque vous utilisez cette syntaxe, les résultats retournés dans les tableaux *tabChamps* et *tabValeurs* sont du type suivant :

tabChamps{1} = "X-METHOD"	*tabValeurs{1} = "GET" **
tabChamps{2} = "X-URL"	*tabValeurs{2} = "/page.html" **
tabChamps{3} = "X-VERSION"	*tabValeurs{3} = "HTTP/1.0" **
tabChamps{4} = "User-Agent"	*tabValeurs{4} = "browser"*
tabChamps{5} = "Cookie"	*tabValeurs{5} = "C=HELLO"*

*Ces trois premiers éléments ne correspondent pas à des champs HTTP. Ils constituent la première ligne de la requête.

Conformément à la norme HTTP, les noms des champs sont toujours libellés en anglais.

A titre indicatif, voici une liste non exhaustive des champs HTTP pouvant être présents dans une requête :

- **Accept** : ce que le navigateur est susceptible d'accepter comme contenu.
- **Accept-Language** : la ou les langue(s) acceptée(s) par le navigateur (pour information). Permet de choisir une page d'accueil en fonction de la langue préférée du navigateur.
- **Cookie** : liste des cookies.
- **From** : adresse e-mail de l'utilisateur du navigateur.
- **Host** : nom ou adresse du serveur (par exemple, dans le cas de l'URL http://monserveurweb/mapage.html, Host prend la valeur "monserveurweb"). Permet de gérer les cas où plusieurs noms pointent vers la même adresse IP (virtual hosting).
- **Referer** : provenance de la requête (par exemple http://monserveurweb/mapage1.html), c'est-à-dire la page que l'utilisateur affiche s'il clique sur le bouton **Précédent** de son navigateur.
- **User-Agent** : nom et version du navigateur ou du proxy.

Exemple

- Cette méthode permet de récupérer le contenu de tout champ d'en-tête de requête HTTP :

```
  ` Méthode projet GetHTTPField
  ` GetHTTPField ( Texte ) -> Texte
  ` GetHTTPField ( Nom en-tête HTTP ) -> Contenu en-tête HTTP
C_TEXTE($0;$1)
C_ENTIER LONG($vlElem)
TABLEAU TEXTE($noms;0)
TABLEAU TEXTE($valeurs;0)
$0:=""
WEB LIRE ENTETE HTTP($noms;$valeurs)
$vlElem:=Chercher dans tableau($noms;$1)
Si($vlElem>0)
   $0:=$valeurs{$vlElem}
Fin de si
```

- Une fois que cette méthode projet est écrite, vous pouvez l'appeler ainsi :

```
  ` Contenu de l'en-tête Cookie
$cookie:=GetHTTPField("Cookie")
```

- Vous pouvez également envoyer des pages différentes en fonction de la langue du navigateur (par exemple dans la **Méthode base Sur connexion**

Web) :

```
$langue:=GetHTTPField("Accept-Language")
Au cas ou
   :($langue="@fr@") `Français (cf. liste ISO 639)
      WEB ENVOYER FICHIER("index_fr.html")
   :($langue="@es@") `Espagnol (cf. liste ISO 639)
      WEB ENVOYER FICHIER("index_es.html")
   Sinon
      WEB ENVOYER FICHIER("index.html")
Fin de cas
```

Note : Les navigateurs Web permettent de définir plusieurs langues par défaut. Elles sont listées dans le champ “Accept-Language”, séparées par des ";". Leur priorité est définie par leur position au sein de la chaîne ; il peut donc être utile de tester la position des langues dans la chaîne.

- Exemple de gestion des hôtes virtuels (par exemple dans la **Méthode base Sur connexion Web**). Les trois noms “home_site.com”, “home_site1.com” et “home_site2.com” pointent vers la même adresse IP, par exemple 192.1.2.3.

```
$host:=GetHTTPField("Host")
Au cas ou
   :($host="www.site1.com")
      WEB ENVOYER FICHIER("home_site1.com")
   :($host="www.site2.com")
      WEB ENVOYER FICHIER("home_site2.com")
   Sinon
      WEB ENVOYER FICHIER("home_site.com")
Fin de cas
```

WEB LIRE EXPIRATION SESSION

WEB LIRE EXPIRATION SESSION (idSession ; dateExp ; heureExp)

Paramètre	Type		Description
idSession	Texte	⇨	UUID de session
dateExp	Date	⇦	Date d'expiration du cookie
heureExp	Heure	⇦	Heure d'expiration du cookie

Description

La commande **WEB LIRE EXPIRATION SESSION** retourne les informations relatives à l'expiration du cookie de la session dont vous avez passé l'UUID dans *idSession*.

Le paramètre *dateExp* reçoit la date d'expiration et le paramètre *heureExp* reçoit l'heure d'expiration du cookie.

Note : À chaque requête Web, la date et l'heure d'expiration du cookie sont réinitialisées à une valeur correspondant à l'instant de la requête+la valeur de l'option Web timeout session. Par exemple :
Première requête, Lundi à 01h00
-> envoi du cookie 4DSID xxxyyy expiration l+24h = Mardi 01:00
Deuxième requête, Lundi à 01h10
-> envoi du cookie 4DSID xxxyyy expiration l+24h = Mardi 01:10
Troisième requête, Mardi à 04h00 : cookie expiré
-> envoi du cookie 4DSID aaabbb expiration l+24h = Mercredi 01:00

WEB Lire ID session courante

WEB Lire ID session courante -> Résultat

Paramètre	Type		Description
Résultat	Texte	➲	UUID de la session

Description

La commande **WEB Lire ID session courante** retourne l'ID de la session ouverte pour la requête Web. Cet ID a été généré automatiquement par 4D sous la forme d'un UUID.

Si cette commande est appelée hors du contexte d'une session Web, elle retourne une chaîne vide "".

WEB Lire nombre parties corps

WEB Lire nombre parties corps -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Nombre de parties du corps

Description

La commande **WEB Lire nombre parties corps** retourne le nombre de parties qui composent le body (corps) reçu.

Exemple

Reportez-vous à l'exemple de la commande **WEB LIRE PARTIE CORPS**.

WEB Lire nombre process session

WEB Lire nombre process session (idSession) -> Résultat

Paramètre	Type		Description
idSession	Texte	➡	UUID de session
Résultat	Entier long	⮌	Nombre de process rattachés à la session

Description

La commande **WEB Lire nombre process session** retourne le nombre de process actifs rattachés à la session dont vous avez passé l'UUID dans le paramètre *idSession*.

Cette commande a été créée dans le contexte de la fonctionnalité **Gestion des sessions 4D Mobile** introduite dans 4D v15 R4. Elle a pour principal but de compter le nombre de process lancés par une session 4D Mobile.

- Pour une session 4D Mobile, cette commande retourne le nombre de process effectivement lancés. Une session 4D Mobile peut piloter plusieurs process.
- Pour une session Web "classique", cette commande retourne toujours 1 (une session Web = un process).

Exemple

Vous souhaitez stocker dans des tableaux les données relatives à la session 4D Mobile courante :

```
C_TEXTE($sessionID)
C_ENTIER LONG($count)
C_DATE($expDate)
C_HEURE($expTime)

$sessionID:=WEB Lire ID session courante
$count:=WEB Lire nombre process session($sessionID)
WEB LIRE EXPIRATION SESSION($sessionID;$expDate;$expTime)

AJOUTER A TABLEAU($aTimestamp;Chaine(Date du jour)+" "+Chaine(Heure courante))
AJOUTER A TABLEAU($aSessionUID;$sessionID)
AJOUTER A TABLEAU($aNbProcesses;$count)
AJOUTER A TABLEAU($aExpirationDate;$expDate)
AJOUTER A TABLEAU($aExpirationTime;$expTime)
```

WEB LIRE OPTION

WEB LIRE OPTION (sélecteur ; valeur)

Paramètre	Type		Description
sélecteur	Entier long	⇨	Code de l'option à modifier
valeur	Entier long, Texte	⇦	Valeur de l'option

Description

La commande **WEB LIRE OPTION** permet de lire la valeur courante d'une option de fonctionnement du serveur Web de 4D.

Le paramètre *sélecteur* contient l'option Web à lire. Passez dans ce paramètre une constante du thème "**Serveur Web**" :

Constante	Type	Valeur	Comment
Web activer validation adresse IP de session	Entier long	83	**Portée** : serveur Web local **Conservé entre deux sessions** : Non **Description** : Permet de désactiver ou d'activer la validation des adresses IP pour les cookies de session. Pour des raisons de sécurité, par défaut le serveur Web de 4D vérifie l'adresse IP de chaque requête contenant un cookie de session et rejette la requête si cette adresse ne correspond pas à l'adresse IP utilisée pour créer le cookie. Dans certaines applications spécifiques, vous pouvez souhaiter désactiver cette validation et accepter les cookies de session même si les adresses IP ne correspondent pas. Par exemple, lorsque les appareils mobiles basculent du réseau Wifi au réseau 3G/4G, leur adresse IP change. Dans ce cas, vous pouvez passer 0 dans cette option afin de permettre aux clients de pouvoir continuer à utiliser leur session Web même après changement d'adresse IP. Notez que ce paramétrage abaisse le niveau de sécurité de votre application. Lorsqu'il est modifié, ce paramétrage est effectif immédiatement (il n'est pas nécessaire de redémarrer le serveur HTTP). **Valeurs possibles** : 0 (désactivé) ou 1 (activé) **Valeur par défaut** : 1 (les adresses IP sont vérifiées)
Web adresse IP d'écoute	Entier long	16	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Adresse IP sur laquelle le serveur Web doit recevoir les requêtes HTTP avec 4D en mode local et 4D Server. Par défaut, aucune adresse particulière n'est spécifiée (*valeur* = 0). Ce paramètre est défini dans les Propriétés de la base. Le sélecteur *Web Adresse IP d'écoute* est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d'accès au mode Développement). Le paramètre *valeur* contient l'adresse IP sous forme hexadécimale. Ainsi, pour désigner une adresse du type "a.b.c.d", le code sera de la forme : `C_ENTIER LONG($addr)` `$addr:=($a<<24)\|($b<<16)|($c<<8)\|$d` `WEB FIXER OPTION(Web Adresse IP découte;$addr)`
Web chemin du cookie de session	Entier long	82	**Portée :** serveur Web local **Conservé entre deux sessions :** Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de fixer ou de lire la valeur du champ "path" du cookie de session. Ce sélecteur (ainsi que le sélecteur 81) est utile pour contrôler la portée des cookies de session : si vous définissez par exemple la valeur "/4DACTION" pour ce sélecteur, le client n'enverra un cookie que pour les requêtes dynamiques débutant par 4DACTION et pas pour les images, pages statiques, etc. **Valeurs :** Texte
Web conserver les sessions	Entier long	70	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet d'activer ou d'inactiver la gestion automatique des sessions (décrit dans la section **Gestion des sessions Web**). **Valeurs** : 1 (activer mode) ou 0 (inactiver mode) **Valeur par défaut** : 1 pour les bases créées depuis la v13, 0 pour les bases converties. A noter que ce mode active également le mécanisme de réutilisation des contextes temporaires en mode distant. Pour plus d'informations sur ce mécanisme, reportez-vous à la description de cette option dans la section **Paramétrages du serveur Web**.
Web debug log	Entier long	84	**Portée :** serveur Web local **Conservé entre deux sessions :** Non, mais reste valide même si le serveur HTTP est redémarré (un nouveau fichier est utilisé) **Description :** Permet de lire ou de définir le statut du fichier de debug de requêtes HTTP du serveur Web 4D. Lorsqu'il est activé, ce fichier, nommé "**HTTPDebugLog_*nn*.txt**", est stocké dans le dossier "Logs" de l'application (*nn* est le numéro de fichier). L'historique des requêtes HTTP est particulièrement utile dans le contexte du débogage du serveur Web. Il stocke en texte brut chaque requête et chaque réponse. La totalité des requêtes, en-têtes compris, est enregistrée ; en option, il est possible d'enregistrer également le corps (body) des requêtes. **Valeurs** : Une des constantes préfixées "wdl" (reportez-vous ci-dessous à la description de ces constantes). **Valeur par défaut** : 0 (non activé)
Web domaine du cookie de session	Entier long	81	**Portée :** serveur Web local **Conservé entre deux sessions :** Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de fixer ou de lire la valeur du champ "domain" du cookie de session. Ce sélecteur (ainsi que le sélecteur 82) est utile pour contrôler la portée des cookies de session : si vous définissez par exemple la valeur "/*.4d.fr" pour ce sélecteur, le client n'enverra un cookie que lorsque la requête s'adresse au domaine ".4d.fr", ce qui exclut les serveurs hébergeant des données statiques externes. **Valeurs :** Texte
			Portée : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Démarrage ou arrêt de l'enregistrement des requêtes Web reçues par le serveur Web de 4D en mode local ou de 4D Server. Par défaut, la valeur est 0 (pas d'enregistrement des requêtes).

Constante	Type	Valeur	Commentaire
Web enreg requêtes	Entier long	29	L'historique des requêtes Web est stocké sous la forme d'un fichier texte nommé "logweb.txt" automatiquement placé dans le dossier Logs de la base, à côté du fichier de structure. Le format de ce fichier est déterminé par la valeur que vous passez. Pour plus d'informations sur les différents formats d'historique des requêtes, reportez-vous à la section **Informations sur le site Web**. L'activation de ce fichier peut également être définie dans la page "Web/Journal" des Propriétés de la base. **Valeurs possibles** : 0 = Ne pas enregistrer (défaut), 1 = Enregistrer au format CLF, 2 = Enregistrer au format DLF, 3 = Enregistrer au format ELF, 4 = Enregistrer au format WLF. **Attention** : Les formats 3 et 4 sont des formats personnalisés, dont le contenu doit être défini au préalable dans les Propriétés de la base. Si vous utilisez l'un de ces formats sans que ses champs n'aient été sélectionnés, le fichier des requêtes n'est pas généré. **Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui
Web jeu de caractères	Entier long	17	**Description** : Jeu de caractères que le serveur Web 4D (avec 4D en mode local et 4D Server) utilise pour communiquer avec les navigateurs Web qui se connectent à la base. La valeur par défaut dépend de la langue du système d'exploitation. Ce paramètre est défini dans les Propriétés de la base. **Valeurs** : Les valeurs possibles dépendent du mode d'exécution de la base relatif au jeu de caractères. • *Mode Unicode* : Lorsque l'application est exécutée en mode Unicode, les valeurs à passer pour ce paramètre sont des identifiants de jeux de caractères (entier long *MIBEnum* ou chaîne *Name*, identifiants définis par l'IANA, cf. http://www.iana.org/assignments/character-sets). Voici la liste des identifiants correspondant aux jeux de caractères pris en charge par le serveur Web de 4D : 4=ISO-8859-1 12=ISO-8859-9 13=ISO-8859-10 17=Shift-JIS 2024=Windows-31J 2026=Big5 38=euc-kr 106=UTF-8 2250=Windows-1250 2251=Windows-1251 2253=Windows-1253 2255=Windows-1255 2256=Windows-1256 • *Mode compatibilité ASCII* : 0 : Occidental 1 : Japonais 2 : Chinois 3 : Coréen 4 : Défini par l'utilisateur 5 : Réservé 6 : Europe Centrale 7 : Cyrillique 8 : Arabe 9 : Grec 10 : Hébreu 11 : Turc 12 : Nordique
Web niveau de compression HTTP	Entier long	50	**Portée** : Serveur Web local **Conservé entre deux sessions** : Non **Description** : Niveau de compression pour tous les échanges HTTP compressés effectués pour le serveur HTTP de 4D (requêtes client ou réponses serveur, Web et Web Service). Ce sélecteur permet d'optimiser les échanges en privilégiant la rapidité d'exécution (compression moindre) ou la quantité de compression (vitesse moindre). Le choix d'une valeur dépend de la taille et de la nature des données échangées. Passez de 1 à 9 dans le paramètre *valeur*, 1 étant la compression la plus rapide et 9 la plus élevée. Vous pouvez également passer -1 pour obtenir un compromis entre vitesse et taux de compression. Par défaut le niveau de compression est 1 (compression rapide). **Valeurs** : 1 à 9 (1 = plus rapide, 9 = plus compressé) ou -1 = meilleur compromis.
Web nom du cookie de session	Entier long	73	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de définir le nom du cookie utilisé pour stocker l'ID de session. **Valeurs :** Texte. **Valeur par défaut :** "4DSID" (passez une chaîne vide pour rétablir la valeur par défaut)
Web nombre de sessions max	Entier long	71	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description** : Permet de limiter le nombre de sessions simultanées. Lorsque le nombre défini est atteint, la session la plus ancienne est détruite (et la **Méthode base Sur fermeture process Web** est appelée) si le serveur Web a besoin d'en créer une nouvelle. **Valeurs** : Entier long. Le nombre de sessions simultanées ne peut pas dépasser le nombre total de process Web (option Web process Web simultanés maxi, 100 par défaut) **Valeur par défaut** : 100 (passez 0 pour rétablir la valeur par défaut)
Web numéro de port HTTPS	Entier long	39	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Numéro du port TCP utilisé par le serveur Web de 4D en mode local et de 4D Server pour les connexions sécurisées via SSL (protocole HTTPS). Le numéro de port HTTPS est défini dans la page "Web/Configuration" de la boîte de dialogue des Propriétés de la base. Par défaut, la valeur est 443 (valeur standard). Vous pouvez utiliser les constantes du thème **Numéros de port TCP** pour le paramètre *valeur*. **Valeurs possibles** : 0 à 65535 Portée : 4D en mode local et 4D Server. **Conservé entre deux sessions :** Non

Constante	Type	Valeur	Comment
Web numéro du port	Entier long	15	**Description :** Permet de fixer ou de lire le numéro du port TCP utilisé par le serveur Web 4D avec 4D en mode local et 4D Server. Le numéro de port TCP est défini dans la page “Web/Configuration” de la boîte de dialogue des Propriétés de la base. Vous pouvez utiliser les constantes du thème **Numéros de port TCP** pour le paramètre *valeur*. Ce sélecteur est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d’accès au mode Développement). **Valeurs :** Pour plus d’informations sur le numéro de port TCP, reportez-vous à la section **Paramétrages du serveur Web**. **Valeur par défaut :** 80
Web process Web simultanés maxi	Entier long	18	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Limite strictement supérieure du nombre de process Web de tout type acceptés par le serveur Web avec 4D en mode local et 4D Server. Lorsque ce nombre limite (moins un) est atteint, 4D ne crée plus de nouveau process et retourne le message “Serveur non disponible” (statut HTTP 503 - Service Unavailable) à toute nouvelle requête. Ce paramètre permet de prévenir la saturation du serveur Web 4D pouvant se produire lors d’un envoi massif de requêtes ou d’une demande excessive de création de contextes. Il peut également être défini dans la boîte de dialogue des Propriétés de la base. En théorie, le nombre maximum de process Web est le résultat de la division Mémoire disponible / Taille de la pile d’un process Web. Une autre solution consiste à visualiser les informations sur les process Web affichées dans l’Exporateur d’exécution : le nombre courant de process Web et le nombre maximum atteint depuis le démarrage du serveur Web sont indiqués. **Valeurs** : Toute valeur incluse entre 10 et 32 000. La valeur par défaut est 100.
Web seuil de compression HTTP	Entier long	51	**Portée** : Serveur HTTP local **Conservé entre deux sessions** : Non **Description** : Dans le cadre d’échanges HTTP optimisés, seuil de taille de requête au-dessous de laquelle les échanges ne doivent pas être compressés. Ce paramétrage est utile pour éviter de perdre du temps machine à compresser les trop petits échanges. **Valeurs possibles** : Toute valeur de type Entier long. Le paramètre *valeur* contient une taille exprimée en octets. Par défaut, le seul de compression est fixé à 1024 octets.
Web taille max requêtes	Entier long	27	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Taille maximale (en octets) des requêtes HTTP entrantes (POST) que le serveur Web est autorisé à traiter. Par défaut, la valeur est 2 000 000, c’est-à-dire un peu moins de 2 Mo. La valeur maximale (2 147 483 648) signifie en pratique qu’aucune limite n’est fixée. Ce paramétrage permet d’empêcher la saturation du serveur Web à cause de requêtes entrantes de trop grande taille. Lorsqu’une requête atteint la limite, le serveur Web de 4D la refuse. **Valeurs possibles** : 500 000 à 2 147 483 648.
Web timeout process	Entier long	78	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de modifier la durée de vie des process inactifs associés aux sessions. A l’issue du timeout, le process est tué sur le serveur, la **Méthode base Sur fermeture process Web** est appelée puis le contexte de la session est détruit. **Valeurs** : Entier long (minutes) **Valeur par défaut** : 480 minutes (passez 0 pour rétablir la valeur par défaut)
Web timeout session	Entier long	72	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description** : Permet de modifier la durée de vie des sessions inactives (durée définie dans le cookie). A l’issue de cette durée, le cookie de session expire et n’est plus envoyé par le client HTTP. **Valeurs** : Entier long (minutes) **Valeur par défaut** : 480 minutes (passez 0 pour rétablir la valeur par défaut)
Web TRACE HTTP	Entier long	85	**Portée** : serveur Web local **Conservé entre deux sessions** : Non **Description** : Permet de désactiver ou d'activer la méthode HTTP TRACE dans le serveur Web de 4D. Pour des raisons de sécurité, à compter de 4D v15 R2, par défaut le serveur Web de 4D Web rejette les requêtes HTTP TRACE avec l'erreur 405 (see Désactivation de HTTP TRACE). Si nécessaire, vous pouvez rétablir la prise en charge de la méthode HTTP TRACE pour la session en passant cette constante avec la valeur 1. Lorsque l'option est activée, le serveur Web de 4D répond aux requêtes HTTP TRACE en retournant la ligne de requête, l'en-tête et le corps. **Valeurs possibles** : 0 (désactivé) ou 1 (activé) **Valeur par défaut** : 0 (désactivé)

Lorsque vous utilisez le *sélecteur* Web debug log, vous pouvez récupérer une des constantes suivantes dans le paramètre *valeur* :

Constante	Type	Valeur	Comment
wdl activer avec body request	Entier long	5	Le fichier de debug de requêtes HTTP Web est activé avec le body de la requête uniquement
wdl activer avec body response	Entier long	3	Le fichier de debug de requêtes HTTP Web est activé avec le body de la réponse uniquement
wdl activer avec tous body	Entier long	7	Le fichier de debug de requêtes HTTP Web est activé avec les body de la requête et de la réponse
wdl activer sans body	Entier long	1	Le fichier de debug de requêtes HTTP Web est activé sans les body (la taille des body est fournie dans ce cas)
wdl désactiver log	Entier long	0	Le fichier de debug de requêtes HTTP Web est désactivé

WEB LIRE PARTIE CORPS

WEB LIRE PARTIE CORPS (partie ; contenuPartie ; nomPartie ; typeMime ; nomFichier)

Paramètre	Type		Description
partie	Entier long	➡	Numéro de partie
contenuPartie	BLOB, Texte	⬅	Contenu de la partie
nomPartie	Texte	⬅	Nom de la variable "input"
typeMime	Texte	⬅	Type mime du fichier
nomFichier	Texte	⬅	Nom du fichier posté

Description

La commande **WEB LIRE PARTIE CORPS**, appelée dans le contexte d'un process Web, permet d'analyser la partie "corps" d'une requête multi-part.

Passez dans le paramètre *partie* le numéro de la partie à analyser. Vous pouvez obtenir le nombre total de parties à l'aide de la commande **WEB Lire nombre parties corps**.

Le paramètre *contenuPartie* récupère le contenu de la partie. Lorsque les parties à récupérer sont des fichiers, vous devez passer un paramètre de type BLOB. Dans le cas de variables TEXT postées dans un formulaire Web, vous pouvez passer un paramètre de type texte.

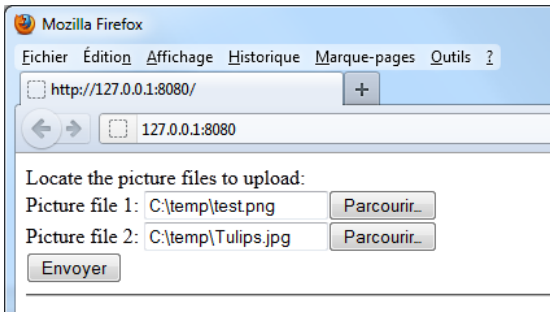
Le paramètre *nomPartie* récupère le nom de la variable du champ input HTTP.

Les paramètres *typeMime* et *nomFichier* permettent de récupérer le type Mime et le nom du fichier d'origine, le cas échéant. *nomFichier* n'est renseigné que dans le cas où le fichier a été posté dans **<input type="file">**.
typeMime et *nomFichier* sont optionnels mais ne peuvent pas être passés séparément.

Note : Dans le cadre d'une requête multi-part, le premier tableau de la commande **WEB LIRE VARIABLES** retourne toutes les parties du formulaire, dans le même ordre que la commande **WEB LIRE PARTIE CORPS**. Vous pouvez l'utiliser par exemple afin d'obtenir directement la position d'une partie du formulaire.

Exemple

Dans cet exemple, un formulaire Web permet de télécharger sur le serveur HTTP plusieurs images depuis un navigateur et de les afficher dans la page.
Voici le formulaire Web :



Voici le code la partie <body> de la page :

```
<body>         <form enctype="multipart/form-data" action="/4DACTION/GetFile/" method="post">             Locate
the picture files to upload: <br>             Picture file 1: <input name="file1" type="file"><br>
Picture file 2: <input name="file2" type="file"><br>             <input type="submit">
</form>             <hr/>     <!--4DSCRIPT/galleryInit-->     <!--4Dloop aFileNames-->         <img
src="/photos/<!--4Dvar aFileNames{aFileNames}-->"/>     <!--4Dendloop--> </body>
```

Deux méthodes 4D sont appelées par la page :

- ***galleryInit*** au chargement (balise 4DSCRIPT), permettant d'afficher les images présentes dans le dossier de destination.
- ***GetFile*** au moment de l'envoi des données (url 4DACTION du formulaire multi-part), permettant de traiter l'envoi.

Voici le code de la méthode ***galleryInit*** :

```
C_TEXTE($vDestinationFolder)
TABLEAU TEXTE(aFileNames;0)
C_ENTIER LONG($i)
$vDestinationFolder:=Dossier 4D(Dossier racine HTML)+"photos"+Séparateur dossier //"DossierWeb/photos" par exemple
LISTE DES DOCUMENTS($vDestinationFolder;aFileNames)
```

Voici le code de la méthode ***GetFile*** :

```
C_TEXTE($vPartName;$vPartMimeType;$vPartFileName;$vDestinationFolder)
C_BLOB($vPartContentBlob)
C_ENTIER LONG($i)
$vDestinationFolder:=Dossier 4D(Dossier racine HTML)+"photos"+Séparateur dossier
Boucle($i;1;WEB Lire nombre parties corps) //pour chaque partie
   WEB LIRE PARTIE CORPS($i;$vPartContentBlob;$vPartName;$vPartMimeType;$vPartFileName)
   Si($vPartFileName#"")
      BLOB VERS DOCUMENT($vDestinationFolder+$vPartFileName;$vPartContentBlob)
   Fin de si
```

```
Fin de boucle
WEB ENVOYER REDIRECTION HTTP("/") // retour à la page
```

WEB LIRE STATISTIQUES

WEB LIRE STATISTIQUES (pages ; hits ; usage)

Paramètre	Type		Description
pages	Tableau texte	⇦	Noms des pages les plus consultées
hits	Tableau entier long	⇦	Nombre de hits pour chaque page
usage	Entier long	⇦	Pourcentages du cache utilisé

Description

La commande **WEB LIRE STATISTIQUES** vous permet d'obtenir des informations sur les pages les plus consultées, chargées dans le cache du serveur Web. Par conséquent, ces statistiques concernent uniquement les pages statiques, les images GIF, les images JPEG <100 ko et les feuilles de style (.css).

Note : Pour plus d'informations sur le paramétrage du cache du serveur Web 4D, reportez-vous à la section **Paramétrages du serveur Web**.

La commande remplit le tableau texte *pages* avec les noms des pages les plus consultées. Le tableau entier long *hits* reçoit le nombre de "hits" pour chaque page. La variable numérique *usage* reçoit le pourcentage du cache Web utilisé par chaque page.

Exemple

Vous souhaitez générer une page semi-dynamique affichant les statistiques d'utilisation du cache Web. Pour cela, dans une page HTML statique appelée "stats.shtm" (les pages suffixées .shtm sont automatiquement analysées par le serveur Web), vous placez la balise *<!—#4DSCRIPT/STATS—>* ainsi que des références aux variables *vPages* et *vUsage*, par exemple :

```
<html> <head><title>Stats Web 4D</title></head> <!--#4DSCRIPT/STATS--> <body> <strong>Pourcentage du cache
utilisé : </strong> <!--#4DTEXT vUsage--> <hr> <strong>Pages les plus consultées : </strong> <!--#4DHTML vPages--
> </body> </html>
```

Dans la méthode projet ***STATS***, écrivez le code suivant :

```
C_TEXTE($1)
C_TEXTE(vPages)
TABLEAU TEXTE(pages;0)
TABLEAU ENTIER LONG(hits;0)
C_ENTIER LONG(vUsage)

STATISTIQUES DU CACHE WEB(pages;hits;vUsage)
Boucle($i;1;Taille tableau(pages))
 // Pour chaque page présente dans le cache
   vPages:=pages{$i}+"   "+Chaine(hits{$i})+"<br>"
 //Insertion du nom de la page et du code HTML
Fin de boucle
```

Vous pouvez envoyer la page "stats.shtm" via un lien URL ou à l'aide de commande **WEB ENVOYER FICHIER**.

WEB LIRE VARIABLES

WEB LIRE VARIABLES (tabNoms ; tabValeurs)

Paramètre	Type		Description
tabNoms	Tableau texte	⇦	Noms des variables du formulaire Web
tabValeurs	Tableau texte	⇦	Valeurs des variables du formulaire Web

Description

La commande **WEB LIRE VARIABLES** remplit les tableaux texte *tabNoms* et *tabValeurs* avec, respectivement, les noms et les valeurs des variables contenues dans un formulaire Web “soumis” (c’est-à-dire envoyé) au serveur Web.
Cette commande récupère la valeur de toutes les variables pouvant être incluses dans des pages HTML : zones de saisie, boutons, cases à cocher, boutons radio, pop up menus, listes d’options...

Note : Dans le cas des cases à cocher, le nom de la variable et sa valeur ne sont retournés que si la case est effectivement cochée.

La commande fonctionne quel que soit le type d’URL ou de formulaire (méthode POST ou GET) envoyé au serveur Web.
Cette commande peut être appelée, selon les besoins, dans la **Méthode base Sur connexion Web** ou toute autre méthode 4D qui résulte de la soumission d’un formulaire.

Précisions sur les formulaires Web et les actions associées

Un formulaire est composé de “zones de saisie” (zones de texte, boutons, cases à cocher), chacune ayant un nom. Lorsqu’un formulaire est “soumis” au serveur Web (une requête est envoyée au serveur), la requête comporte, entre autres, la liste des zones de saisie et leurs valeurs respectives.
Il y a deux “méthodes” pour soumettre un formulaire (4D accepte indifféremment l’une ou l’autre) :

- POST, généralement utilisée pour l’insertion de données dans le serveur Web — vers une base de données,
- GET, généralement utilisée pour l’interrogation du serveur Web — données en provenance d’une base de données.

Exemple

Un formulaire contient deux champs, vNOM et vVILLE, qui reçoivent les valeurs “MARTIN” et “PARIS”. L’action associée au formulaire est “*/4DACTION/WEBFORM*”.

- Si la méthode du formulaire est POST (cas le plus souvent utilisé), les données saisies ne seront pas visibles dans l’URL (c’est-à-dire http://127.0.0.1/4DACTION/WEBFORM).
- Si la méthode du formulaire est GET, les données seront visibles dans l’URL (c’est-à-dire http://127.0.0.1/4DACTION/WEBFORM?vNOM=MARTIN&vVILLE=PARIS).

La méthode WEBFORM peut être de la forme suivante :

```
TABLEAU TEXTE($tnoms;0)
TABLEAU TEXTE($tvaleurs;0)
WEB LIRE VARIABLES($tnoms;$tvaleurs)
```

On obtient alors :

```
$tnoms{1}="vNOM"
$tnoms{2}="vVILLE"
$tvaleurs{1}="MARTIN"
$tvaleurs{2}="PARIS"
```

WEB Serveur est démarré

WEB Serveur est démarré -> Résultat

Paramètre	Type		Description
Résultat	Booléen	➲	Vrai si le serveur Web est démarré, sinon Faux

Description

La commande **WEB Serveur est démarré** retourne **Vrai** si le serveur Web intégré de 4D est démarré, et **Faux** si le serveur Web est stoppé.
La commande retourne le statut du serveur Web de la machine sur laquelle elle est exécutée :

Contexte d'exécution	Retourne le statut de
4D monoposte	Serveur Web 4D local
4D Server	Serveur Web 4D Server
4D mode distant (process local)	Serveur Web 4D local
4D mode distant (procédure stockée sur 4D Server)	Serveur Web 4D Server
4D mode distant (procédure stockée sur une autre 4D distant)	Serveur Web 4D distant

Exemple

Vous souhaitez tester si le serveur Web de 4D est lancé :

```
Si(WEB Serveur est démarré)
   ... //effectuer les actions appropriées
Fin de si
```

WEB Valider digest

WEB Valider digest (nomUtilisateur ; motDePasse) -> Résultat

Paramètre	Type		Description
nomUtilisateur	Texte	⇒	Nom de l'utilisateur
motDePasse	Texte	⇒	Mot de passe de l'utilisateur
Résultat	Booléen	⇐	Vrai=Authentification correcte, Faux=Echec de l'authentification

Description

La commande **WEB Valider digest** permet de vérifier la validité des identifiants (nom et mot de passe) fournis par un utilisateur se connectant au serveur Web. Cette commande doit être utilisée dans la **Méthode base Sur authentification Web** dans le cadre d'une authentification Web en mode Digest (cf. section **Sécurité des connexions**).

Passez dans les paramètres *nomUtilisateur* et *motDePasse* les identifiants de l'utilisateur conservés en local. La commande utilise ces identifiants pour générer une valeur qu'elle compare aux informations envoyées par le navigateur Web.

Si les valeurs sont identiques, la commande retourne Vrai. Sinon, elle retourne Faux.

Ce mécanisme vous permet de gérer et de maintenir par programmation votre propre système sécurisé d'accès au serveur Web. A noter que la validation Digest ne peut pas être utilisée conjointement avec les mots de passe 4D.

Note : Si le navigateur ne prend pas en charge l'authentification Digest, une erreur est retournée (erreur d'authentification).

Exemple

Exemple de **Méthode base Sur authentification Web** en mode Digest

```
 // Méthode base Sur authentification Web
C_TEXTE($1;$2;$5;$6;$3;$4)
C_TEXTE($utilisateur)
C_BOOLEEN($0)
$0:=Faux
$utilisateur:=$5
    //Pour des raisons de sécurité, refuser les noms qui contiennent @
Si(AvecJoker($utilisateur))
      $0:=Faux
          //La méthode AvecJoker est décrite dans la section "Méthode base Sur authentification Web"
Sinon
      CHERCHER([WebUsers];[WebUsers]User=$utilisateur)
      Si(OK=1)
         $0:=WEB Valider digest($utilisateur;[WebUsers]Mdp)
      Sinon
         $0:=Faux //Utilisateur inexistant
      Fin de si
Fin de si
```

_o_Contexte Web

_o_Contexte Web -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Retourne toujours Faux

Commande désactivée

Cette commande est désactivée à compter de 4D v13. Si elle est appelée, l'erreur 33 ("Méthode ou fonction non implémentée") est générée.

_o_FIXER EXECUTABLE CGI

_o_FIXER EXECUTABLE CGI (url1 {; url2})

Paramètre	Type		Description
url1	Chaîne	⇒	*** inutilisé
url2	Chaîne	⇒	*** inutilisé

Commande désactivée

Cette commande est désactivée à compter de 4D v13. Si elle est appelée, l'erreur 33 ("Méthode ou fonction non implémentée") est générée.

_o_FIXER LIMITES AFFICHAGE WEB

_o_FIXER LIMITES AFFICHAGE WEB (nombreEnr {; nombrePages {; refImage}})

Paramètre	Type		Description
nombreEnr	Entier long	⇨	*** inutilisé
nombrePages	Entier long	⇨	*** inutilisé
refImage	Entier long	⇨	*** inutilisé

Commande désactivée

Cette commande est désactivée à compter de 4D v13. Si elle est appelée, l'erreur 33 ("Méthode ou fonction non implémentée") est générée.

_o_FIXER TEMPORISATION WEB

_o_FIXER TEMPORISATION WEB (timeout)

Paramètre	Type		Description
timeout	Entier long	⇨	*** inutilisé

Commande désactivée

Cette commande est désactivée à compter de 4D v13. Si elle est appelée, l'erreur 33 ("Méthode ou fonction non implémentée") est générée.

Sous-enregistrements

- Lire cle sous enregistrement
- *_o_ALLER À DERNIER SOUS ENREGISTREMENT*
- *_o_APPLIQUER À SOUS SÉLECTION*
- *_o_Avant sous enregistrement*
- *_o_CHERCHER SOUS ENREGISTREMENTS*
- *_o_CRÉER SOUS ENREGISTREMENT*
- *_o_DÉBUT SOUS ENREGISTREMENT*
- *_o_Fin sous enregistrement*
- *_o_SOUS ENREGISTREMENT PRECEDENT*
- *_o_SOUS ENREGISTREMENT SUIVANT*
- *_o_Sous enregistrements trouvés*
- *_o_SUPPRIMER SOUS ENREGISTREMENT*
- *_o_TOUS LES SOUS ENREGISTREMENTS*
- *_o_TRIER SOUS ENREGISTREMENTS*

Lire cle sous enregistrement

Lire cle sous enregistrement (champID) -> Entier long

Paramètre	Type		Description
champID	Champ	⇒	Champ de type "Lien sous-table" ou de type "Entier long" d'une ancienne relation sous-table
Entier long	Entier long	⇐	Clé interne du lien

Description

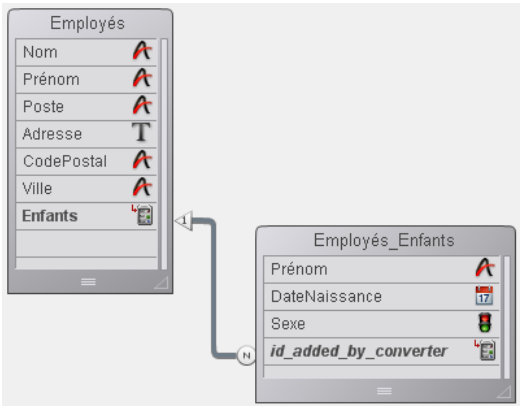
La commande **Lire cle sous enregistrement** est destinée à faciliter la migration du code 4D utilisant des sous-tables converties vers le code standard de manipulation des tables.

Rappel : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Lors de la conversion d'une ancienne base, les sous-tables existantes sont automatiquement transformées en tables standard reliées aux tables d'origine par un lien automatique. La sous-table devient une table "N" et la table d'origine la table "1". Dans la table 1, l'ex-champ de type sous-table est transformé en champ spécial de type "Lien sous-table" et dans la table N, un champ spécial, également de type "Lien sous-table" est ajouté. Il est nommé "id_added_by_converter".

Ce principe permet de préserver le fonctionnement des bases de données converties, mais il est fortement conseillé de remplacer les mécanismes des sous-tables par ceux des tables standard.

La première étape de ce processus consiste à supprimer le lien automatique spécial, ce qui désactive définitivement les mécanismes hérités des sous-tables. Cette opération nécessite ensuite de réécrire le code associé. La commande **Lire cle sous enregistrement** accompagne cette réécriture en renvoyant l'identifiant interne utilisé par le lien. Elle permet de s'affranchir du lien et donc de travailler avec la sélection de l'ancienne sous-table, que le lien soit présent ou non.

Examinons par exemple la structure (convertie) suivante :



Dans 4D, le code suivant continue de fonctionner mais il doit être mis à jour :

```
TOUS LES SOUS ENREGISTREMENTS([Employés]Enfants)
$total:=Sous enregistrements trouves([Employés]Enfants)
vPrenoms:=""
Boucle($i;1;$total)
   vPrenoms:=vPrenoms+[Employés]Enfants'Prénom+" "
   SOUS ENREGISTREMENT SUIVANT([Employés]Enfants)
Fin de boucle
```

Vous pouvez désormais remplacer ce code par :

```
CHERCHER([Employés_Enfants];[Employés_Enfants]id_added_by_converter=Lire cle sous
enregistrement([Employés]Enfants))
$total:=Enregistrements trouves([Employés_Enfants])
vPrenoms:=""
Boucle($i;1;$total)
   vPrenoms:=vPrenoms+[Employés_Enfants]Prénom+" "
   ENREGISTREMENT SUIVANT(Employés_Enfants)
Fin de boucle
```

Note : S'il n'y a pas d'enregistrement courant chargé au moment de son exécution, **Lire cle sous enregistrement** retourne 0.

Le second code présente le double avantage d'utiliser des commandes standard de 4D et de fonctionner de manière identique, que le lien soit présent ou non. Lorsque vous supprimerez le lien, la commande retournera simplement la valeur clé stockée dans le champ Entier long.

La commande accepte dans le paramètre *champ_ID* soit un champ de type Lien sous-table (lien existant) soit de type Entier long (lien supprimé). Dans tous les autres cas, une erreur est générée.

Ce principe permet d'écrire du code de transition. Lors de la dernière étape de la mise à niveau de l'application, vous pourrez supprimer les appels à cette commande.

Affecter le champ id_added_by_converter

A compter de 4D v14 R3, vous pouvez affecter la valeur du champ "id_added_by_converter". Jusqu'alors, cette valeur pouvait uniquement être affectée par 4D, ce qui obligeait les développeurs à utiliser des commandes obsolètes telles que **_o_CRÉER SOUS ENREGISTREMENT** pour pouvoir ajouter des enregistrements dans les sous-tables converties.

Avec cette possibilité, vous pouvez convertir vos anciennes bases comportant des sous-tables d'une manière progressive : vous pouvez conserver le lien

spécial "Lien sous-table", tout en ajoutant ou en modifiant des enregistrements liés comme s'ils étaient standard. Une fois que toutes vos méthodes auront été mises à jour, vous pourrez remplacer le lien spécial par un lien normal sans changer votre code.

Par exemple, vous pouvez écrire avec la structure précédente :

```
CREER ENREGISTREMENT([Employés])
[Employés]Nom:="Jones"
CREER ENREGISTREMENT([Employés_Enfants])
[Employés_Enfants]Prénom:="Natacha"
[Employés_Enfants]DateNaissance:=!12/24/2013!
[Employés_Enfants]id_added_by_converter:=Lire cle sous enregistrement([Employés]Enfants)
STOCKER ENREGISTREMENT([Employés_Enfants])
STOCKER ENREGISTREMENT([Employés]
```

Ce code fonctionnera indifféremment avec un lien spécial ou standard.

_o_ALLER À DERNIER SOUS ENREGISTREMENT

_o_ALLER À DERNIER SOUS ENREGISTREMENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table dans laquelle se placer sur le dernier sous-enregistrement

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_APPLIQUER À SOUS SÉLECTION

_o_APPLIQUER À SOUS SÉLECTION (sousTable ; formule)

Paramètre	Type		Description
sousTable	Sous-table	⇒	Sous-table à laquelle appliquer la formule
formule	Instruction	⇒	Ligne de code ou méthode

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_Avant sous enregistrement

_o_Avant sous enregistrement (sousTable) -> Résultat

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table pour laquelle vous testez si le pointeur se trouve avant la sous-sélection
Résultat	Booléen	⇦	Avant la sous-sélection (Vrai), sinon (Faux)

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_CHERCHER SOUS ENREGISTREMENTS

_o_CHERCHER SOUS ENREGISTREMENTS (sousTable ; formule)

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table dans laquelle effectuer une recherche
formule	Booléen	⇨	Formule de recherche

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_CRÉER SOUS ENREGISTREMENT

_o_CRÉER SOUS ENREGISTREMENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table dans laquelle vous voulez créer un sous-enregistrement

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_DÉBUT SOUS ENREGISTREMENT

_o_DÉBUT SOUS ENREGISTREMENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table de laquelle charger le premier sous-enregistrement de la sélection courante

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_Fin sous enregistrement

_o_Fin sous enregistrement (sousTable) -> Résultat

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table pour laquelle tester si le pointeur de sous-enregistrement courant est placé au-delà du dernier sous-enregistrement sélectionné
Résultat	Booléen	⇦	Oui (Vrai) ou Non (Faux)

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_SOUS ENREGISTREMENT PRECEDENT

_o_SOUS ENREGISTREMENT PRECEDENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table dans laquelle se placer sur le sous- enregistrement précédent de la sélection courante

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_SOUS ENREGISTREMENT SUIVANT

_o_SOUS ENREGISTREMENT SUIVANT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table dans laquelle se placer sur le sous- enregistrement suivant de la sélection courante

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_Sous enregistrements trouvés

_o_Sous enregistrements trouvés (sousTable) -> Résultat

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table dont vous voulez connaître le nombre de sous-enregistrements
Résultat	Entier long	⇦	Nombre de sous-enregistrements de la sous-sélection courante

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_SUPPRIMER SOUS ENREGISTREMENT

_o_SUPPRIMER SOUS ENREGISTREMENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table de laquelle supprimer le sous-enregistrement courant

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_TOUS LES SOUS ENREGISTREMENTS

_o_TOUS LES SOUS ENREGISTREMENTS (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	⇨	Sous-table dans laquelle sélectionner tous les sous-enregistrements

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

_o_TRIER SOUS ENREGISTREMENTS

_o_TRIER SOUS ENREGISTREMENTS (sousTable ; sousChamp {; > ou <}{; sousChamp2 ; > ou <2 ; ... ; sousChampN ; > ou <N})

Paramètre	Type		Description
sousTable	Sous-table	⇒	Sous-table contenant le(s) sous-champ(s) à trier
sousChamp	Sous-champ	⇒	Sous-champ sur lequel effectuer le tri
> ou <	Opérateur	⇒	> tri croissant ou < tri décroissant

Note de compatibilité

Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

SQL

- Présentation des commandes du thème SQL
- Méthode base Sur authentification SQL
- ARRETER SERVEUR SQL
- CHERCHER PAR SQL
- Debut SQL
- Fin SQL
- FIXER VALEUR CHAMP NULL
- LANCER SERVEUR SQL
- Lire source donnees courante
- LISTE SOURCES DONNEES
- SQL ANNULER CHARGEMENT
- SQL CHARGER ENREGISTREMENT
- SQL EXECUTER
- SQL EXECUTER SCRIPT
- SQL EXPORTER BASE
- SQL EXPORTER SELECTION
- SQL Fin de selection
- SQL FIXER OPTION
- SQL FIXER PARAMETRE
- SQL LIRE DERNIERE ERREUR
- SQL LIRE OPTION
- SQL LOGIN
- SQL LOGOUT
- Valeur champ Null
- *_o_UTILISER BASE EXTERNE*
- *_o_UTILISER BASE INTERNE*

Présentation des commandes du thème SQL

4D comporte un moteur SQL intégré. Le programme inclut également un serveur SQL pouvant répondre aux requêtes d'autres applications 4D ou d'applications tierces (via le pilote ODBC de 4D).

La documentation SQL de 4D se compose de deux parties :

- le **Guide de référence SQL de 4D** (**4D - Référence SQL**). Ce manuel décrit les différents modes d'accès au moteur SQL de 4D, le paramétrage du serveur SQL ainsi que les commandes et mots-clés utilisables dans les requêtes SQL (par exemple **SELECT** ou **UPDATE**). Reportez-vous à ce manuel pour plus d'informations sur l'utilisation du langage SQL dans 4D.
- le **thème SQL du manuel "Langage" de 4D** (**SQL**). Ce thème regroupe les diverses commandes internes de 4D relatives à l'exploitation du SQL dans 4D :
  - contrôle du serveur SQL : **LANCER SERVEUR SQL** et **ARRETER SERVEUR SQL**.
  - accès direct au moteur SQL intégré : **FIXER VALEUR CHAMP NULL**, **Valeur champ Null**, **CHERCHER PAR SQL**.
  - gestion des connexions aux sources de données internes ou externes (SQL Pass-through) : **LISTE SOURCES DONNEES**, **Lire source donnees courante**, **SQL LOGIN**, **SQL LOGOUT**.
  - commandes de haut niveau pour la manipulation des données dans le cadre de connexions SQL directes ou via ODBC : **Debut SQL**, **Fin SQL**, **SQL ANNULER CHARGEMENT**, **SQL CHARGER ENREGISTREMENT**, **SQL EXECUTER**, **SQL Fin de selection**, **SQL FIXER OPTION**, **SQL FIXER PARAMETRE**, **SQL LIRE DERNIERE ERREUR**, **SQL LIRE OPTION**..

Principes de fonctionnement des commandes SQL de haut niveau

Les commandes SQL intégrées de 4D débutent par le préfixe "SQL". Elles appliquent les principes suivants :

- Sauf indication contraire, vous pouvez utiliser ces commandes avec le moteur SQL interne de 4D ou dans une connexion externe ouverte directement ou via ODBC. La commande **SQL LOGIN** vous permet de définir le type de connexion à ouvrir.
- La portée d'une connexion est le process. Si vous souhaitez gérer plusieurs connexions simultanément, vous devez démarrer un process par **SQL LOGIN**.
  La commande **SQL ANNULER CHARGEMENT** permet d'exécuter plusieurs requêtes **SELECT** dans la même connexion.
- Vous pouvez intercepter les erreurs ODBC éventuellement générées lors de l'exécution d'une des commandes SQL de haut niveau à l'aide la commande **APPELER SUR ERREUR**. La commande **SQL LIRE DERNIERE ERREUR** permet dans ce cas d'obtenir des informations supplémentaires.

Prise en charge du standard ODBC

Le standard ODBC (Open DataBase Connectivity) définit une librairie de fonctions standardisées. Ces fonctions permettent à une application telle que 4D d'accéder via le langage SQL à tout système de gestion de données compatible ODBC (bases de données, tableurs, autre application 4D, etc.).

Note : 4D permet également d'importer et d'exporter des données dans une source ODBC via les commandes **IMPORTER ODBC** et **EXPORTER ODBC** ou "manuellement" en mode Développement. Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.

Note : Les commandes SQL de haut niveau de 4D permettent de mettre en place des solutions simples pour faire communiquer les applications 4D et des sources de données ODBC. Si vos applications nécessitent une prise en charge plus étendue du standard ODBC, vous devrez acquérir le plug-in ODBC "bas niveau" de 4D, **4D ODBC Pro**.

Correspondance des types de données

Le tableau suivant liste les correspondances établies automatiquement par 4D entre les types de données 4D et SQL :

Type 4D	Type SQL
C_ALPHA	SQL_C_CHAR
C_TEXTE	SQL_C_CHAR
C_REEL	SQL_C_DOUBLE
C_DATE	SQL_C_TYPE_DATE
C_HEURE	SQL_C_TYPE_TIME
C_BOOLEEN	SQL_C_BIT
C_ENTIER	SQL_C_SHORT
C_ENTIER LONG	SQL_C_SLONG
C_BLOB	SQL_C_BINARY
C_IMAGE	SQL_C_BINARY
C_GRAPHE	SQL_C_BINARY

Note : Les données de type objet (**C_OBJET**) ne sont pas prises en charge par le moteur SQL de 4D.

Référencer des expressions 4D dans les requêtes SQL

4D propose deux modes d'insertion des expressions 4D (variable, tableau, champ, pointeur, expression valide) dans les requêtes SQL : l'association directe et la définition de paramètres via **SQL FIXER PARAMETRE**.

L'association directe peut être effectuée de deux manières :

- insérer le nom de l'objet 4D entre les caractères **<<** et **>>** dans le texte de la requête.
- faire précéder la référence de deux-point ":"
  Exemples :

```
SQL EXECUTER("INSERT INTO emp (empno,ename) VALUES (<<vEmpno>>,<<vEname>>)")
SQL EXECUTER("SELECT age FROM People WHERE name= :vNom")
```

Note de compatibilité : Jusqu'à la version 11.6 de 4D, en mode compilé, vous ne pouviez pas utiliser de références à des variables locales (débutant par le signe $). Cette limitation est supprimée dans 4D à compter de la version 11.7.

Dans ces exemples, les valeurs courantes des variables 4D vEmpno, vEname et vNom seront substituées aux paramètres lors de l'exécution de la requête. Cette solution fonctionne également avec les champs et les tableaux 4D.

Cette syntaxe, simple d'utilisation, présente toutefois l'inconvénient de n'être pas conforme à la norme SQL et de ne pas permettre l'utilisation de paramètres de sortie. Pour y remédier, vous pouvez utiliser la commande **SQL FIXER PARAMETRE**. Cette commande permet de définir chaque objet 4D à intégrer dans une requête ainsi que son mode d'utilisation (entrée, sortie ou les deux). La syntaxe produite est alors standard. Pour plus d'informations, reportez-vous à la description de la commande **SQL FIXER PARAMETRE**.

(1) Cet exemple exécute une requête SQL utilisant directement des tableaux 4D associés :

```
TABLEAU TEXTE(MonTabTexte;10)
TABLEAU ENTIER LONG(MonTabLong;10)

Boucle(vCounter;1;Taille tableau(MonTabTexte))
   MonTabTexte{vCounter}:="Texte"+Chaine(vCounter)
   MonTabLong{vCounter}:=vCounter
Fin de boucle
SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<MonTabTexte>>, <<MonTabLong>>)"
SQL EXECUTER(SQLStmt)
```

(2) Cet exemple permet d'exécuter une requête SQL utilisant directement des champs 4D associés :

```
TOUT SELECTIONNER([Table 2])
SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<[Table 2]Champ1>"+">,<<[Table
2]Champ2>>)"
SQL EXECUTER(SQLStmt)
```

(3) Cet exemple permet d'exécuter une requête SQL passant directement une variable via un pointeur non dépointé :

```
C_ENTIER LONG($vLong)
C_POINTEUR($vPointeur)
$vLong:=1
$vPointeur:=->$vLong
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT Col1 FROM TEST WHERE Col1=:$vPointeur"
SQL EXECUTER(SQLStmt)
```

Utilisation de variables locales en mode compilé

En mode compilé, vous pouvez utiliser des références de variables locales (débutant par le caractère $) dans les instructions SQL sous certaines conditions :

- vous pouvez utiliser des variables locales à l'intérieur d'une séquence **Debut SQL** / **Fin SQL**, hormis avec la commande **EXECUTE IMMEDIATE** ;
- vous pouvez utiliser des variables locales avec la commande **SQL EXECUTER** lorsque ces variables sont utilisées directement dans le paramètre de requête SQL et non via des références.
  Par exemple, le code suivant fonctionnera en mode compilé :

  ```
  SQL EXECUTER("select * from t1 into :$mavar") // fonctionne en mode compilé
  ```

  Le code suivant générera une erreur en mode compilé :

  ```
  C_TEXTE(tRequete)
  tRequete:="select * from t1 into :$mavar"
  SQL EXECUTER(tRequete) // erreur en mode compilé
  ```

Récupération des valeurs dans 4D

La récupération dans le langage de 4D des valeurs issues de requêtes SQL peut être effectuée de deux manières :

- en utilisant les paramètres supplémentaires de la commande **SQL EXECUTER** (solution conseillée).
- en utilisant la clause INTO dans la requête SQL elle-même (solution à réserver aux cas particuliers).

Affichage du résultat d'une requête SQL dans une List box

Il est possible de placer directement le résultat d'une requête SQL dans une list box de type tableau. Cette fonction offre un moyen rapide de visualiser le résultat des requêtes SQL. Seules les requêtes de type **SELECT** peuvent être utilisées. Ce mécanisme n'est pas utilisable avec une base SQL externe.

Les principes de mise en oeuvre sont les suivants :

- Vous créez la list box devant recevoir le résultat de la requête. La source de données de la list box doit être **Tableaux**.
- Vous exécutez la requête SQL de type **SELECT** et assignez le résultat à la variable associée la list box. Vous pouvez utiliser les mots-clés **Debut SQL**/**Fin SQL** (cf. manuel *Langage* de 4D).
- Les colonnes de la list box sont triables et modifiables par l'utilisateur.

- Chaque nouvelle exécution d'une requête **SELECT** avec la list box provoque la réinitialisation des colonnes (il n'est pas possible de remplir progressivement une même list box à l'aide de plusieurs requêtes **SELECT**).
- Il est préférable de placer dans la list box autant de colonnes qu'il y aura de colonnes SQL dans le résultat de la requête SQL. Si le nombre de colonnes de la list box est inférieur à celui requis par la requête **SELECT**, des colonnes sont automatiquement ajoutées. Si le nombre de colonnes de la list box est supérieur à celui requis par la requête **SELECT**, les colonnes superflues sont automatiquement masquées.
  Note : Les colonnes ajoutées automatiquement sont liées à des **Variables dynamiques** de type tableau. La durée de vie de ces tableaux dynamiques est celle du formulaire. Une variable dynamique est également créée pour chaque en-tête. Lorsque la commande **LISTBOX LIRE TABLEAUX** est appelée, le paramètre *tabVarCols* contient des pointeurs vers les tableaux dynamiques et le paramètre *tabVarEntêtes* contient des pointeurs vers les variables d'entête dynamiques. Si une colonne ajoutée est par exemple la cinquième, son nom est *sql_column5* et son nom d'en-tête *sql_header5*.
- En mode interprété, les tableaux existants et utilisés pourront être retypés automatiquement en fonction des données renvoyées par la requête SQL.

Exemple

Nous voulons récupérer tous les champs de la table PERSONS et placer leur contenu dans la list box dont le nom de variable est *vlistbox*. Dans la méthode objet d'un bouton (par exemple), il suffit d'écrire :

```
Debut SQL
   SELECT * FROM PERSONS INTO <<vlistbox>>
Fin SQL
```

Méthode base Sur authentification SQL

La **Méthode base Sur authentification SQL** permet de filtrer les requêtes adressées au serveur SQL intégré de 4D. Le filtrage peut être effectué sur la base du nom, du mot de passe ainsi que (facultativement) de l'adresse IP de l'utilisateur. Le développeur peut utiliser sa propre table d'utilisateurs ou celle des utilisateurs 4D pour évaluer les identifiants de connexion. Une fois la connexion authentifiée, la commande **CHANGER UTILISATEUR COURANT** doit être appelée afin de contrôler les accès de la requête au sein de la base 4D.

Lorsqu'elle existe, la **Méthode base Sur authentification SQL** est automatiquement appelée par 4D ou 4D Server à chaque connexion externe au serveur SQL. Le système interne de gestion des utilisateurs de 4D n'est alors pas sollicité. La connexion n'est acceptée que si la méthode base retourne **Vrai** dans $0 et si la commande **CHANGER UTILISATEUR COURANT** a été exécutée avec succès. Si l'une des deux conditions n'est pas remplie, la requête est rejetée.

Note : L'instruction **SQL LOGIN(SQL_INTERNAL;$utilisateur;$motdepasse)** ne déclenche pas l'appel de la **Méthode base Sur authentification SQL** car il s'agit dans ce cas d'une connexion interne.

La méthode base reçoit jusqu'à trois paramètres de type Texte, passés par 4D ($1, $2 et $3), et retourne un booléen, $0. Voici la description de ces paramètres :

Paramètres	Type	Description
$1	Texte	Nom d'utilisateur
$2	Texte	Mot de passe
$3	Texte	(Facultatif) Adresse IP du client à l'origine de la requête(*)
$0	Booléen	Vrai = requête acceptée, Faux = requête rejetée

(*) 4D retourne les adresses IPv4 dans un format hybride IPv6 comprenant un préfixe de 96 bits, par exemple ::ffff:192.168.2.34 pour l'adresse IPv4 192.168.2.34. Pour plus d'informations, reportez-vous à la section **Prise en charge d'IP v6**.

Vous devez déclarer ces paramètres de la manière suivante :

```
  ` Méthode base Sur authentification Web
C_TEXTE($1;$2;$3)
C_BOOLEEN($0)
... ` Code pour la méthode
```

Le mot de passe ($2) est reçu en texte standard.

Vous devez contrôler les identifiants de la connexion SQL dans la **Méthode base Sur authentification SQL**. Par exemple, vous pouvez contrôler le nom et le mot de passe à l'aide d'une table d'utilisateurs personnalisée. Si les identifiants sont valides, passez Vrai dans $0. Sinon, passez **Faux** dans $0, dans ce cas la connexion est rejetée.

Si vous avez passé **Vrai** dans $0, vous devez ensuite appeler avec succès la commande **CHANGER UTILISATEUR COURANT** dans la **Méthode base Sur authentification SQL** pour que la requête soit acceptée et que 4D ouvre une session SQL pour l'utilisateur.
L'utilisation de la commande **CHANGER UTILISATEUR COURANT** permet de mettre en place un système d'authentification virtuelle ayant comme double avantage le contrôle des actions au sein de la connexion et le masquage pour l'extérieur des identifiants de la connexion dans la session SQL 4D.

Note : Si la **Méthode base Sur authentification SQL** n'existe pas, la connexion est évaluée à l'aide du système intégré de gestion des utilisateurs de 4D s'il est actif, c'est-à-dire si un mot de passe a été attribué au Super_Utilisateur. Si le système n'est pas actif, les utilisateurs sont connectés avec les droits du Super_Utilisateur (accès libre).

Cet exemple de **Méthode base Sur authentification SQL** vérifie que la demande de connexion provient du réseau interne, valide les identifiants puis affecte les droits d'utilisateur "sql_user" pour la session SQL.

```
C_TEXTE($1;$2;$3)
C_BOOLEEN($0)
  `$1 : utilisateur
  `$2 : mot de passe
  `{$3 : Adresse IP du client}
APPELER SUR ERREUR("SQL_error")
Si(checkInternalIP($3))
  `La méthode checkInternalIP vérifie que l'adresse IP est interne
   Si($1="victor") & ($2="hugo")
      CHANGER UTILISATEUR COURANT("sql_user";"")
      Si(OK=1)
         $0:=Vrai
      Sinon
         $0:=Faux
      Fin de si
   Sinon
      $0:=Faux
   Fin de si
Sinon
   $0:=Faux
Fin de si
```

ARRETER SERVEUR SQL

ARRETER SERVEUR SQL
Ne requiert pas de paramètre

Description

La commande **ARRETER SERVEUR SQL** stoppe le serveur SQL intégré de l'application 4D sur laquelle elle a été exécutée. Si le serveur SQL était lancé, toutes les connexions SQL sont interrompues et le serveur n'accepte plus aucune requête SQL externe. Si le serveur SQL n'était pas lancé, la commande ne fait rien.

Note : Cette commande n'influe pas sur le fonctionnement du moteur SQL interne de 4D. Le moteur SQL toujours disponible pour les requêtes internes.

CHERCHER PAR SQL

CHERCHER PAR SQL ({laTable ;} formuleSQL)

Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle retourner une sélection d'enregistrements ou Table par défaut si ce paramètre est omis
formuleSQL	Chaîne	⇒	Formule de recherche SQL valide représentant la clause WHERE de la requête SELECT

Description

La commande **CHERCHER PAR SQL** permet de tirer directement parti du moteur SQL intégré de 4D. Elle exécute une requête SELECT simple qui peut être écrite ainsi :

```
SELECT *
   FROM laTable
   WHERE <formuleSQL>
```

laTable est le nom de la table passé en premier paramètre et *formuleSQL* la chaîne de recherche passée en deuxième paramètre.

Par exemple, l'instruction suivante :

```
CHERCHER PAR SQL([Employees];"name='smith'")
```

équivaut à la requête SQL :

```
SELECT * FROM Employees WHERE "name='smith'"
```

La commande **CHERCHER PAR SQL** est semblable à la commande **CHERCHER PAR FORMULE**. Elle effectue une recherche parmi les enregistrements de la table définie. Elle modifie la sélection courante de *table* pour le process courant et fait du premier enregistrement de la nouvelle sélection le nouvel enregistrement courant.

Note : La commande **CHERCHER PAR SQL** ne peut pas être utilisée dans le contexte d'une connexion SQL externe, elle s'adresse directement au moteur SQL intégré de 4D.

CHERCHER PAR SQL applique *formuleSQL* à chaque enregistrement de la sélection de la table. *formuleSQL* est une expression booléenne qui doit retourner **VRAI** ou **FAUX**. Comme vous le savez peut-être, dans la norme SQL, une condition de recherche peut avoir un résultat **VRAI**, **FAUX** ou NULL. Tous les enregistrements (rows) pour lesquels la condition de recherche retourne **VRAI** sont inclus dans la nouvelle sélection courante.

L'expression *formuleSQL* peut être simple, comme par exemple la comparaison d'un champ (colonne) à une valeur ; elle peut également être complexe, comme la réalisation d'un calcul. Comme **CHERCHER PAR FORMULE**, **CHERCHER PAR SQL** peut évaluer des valeurs dans les tables liées (cf. exemple 4). *formuleSQL* doit être une instruction SQL valide, conforme à la norme SQL-2 et tenant compte de l'implémentation actuelle du SQL dans 4D. Pour plus d'information la prise en charge du SQL dans 4D, reportez-vous au manuel Guide de référence 4D SQL.

Le paramètre *formuleSQL* peut contenir des références à des expressions 4D. La syntaxe à utiliser est la même que pour les commandes SQL intégrées ou le code inclus dans les balises **Debut SQL**/**Fin SQL**, c'est-à-dire : *<<MaVar>>* ou *:MaVar*
Pour plus d'informations sur ce point, reportez-vous à la section **Présentation des commandes du thème SQL**.

Note : Cette commande est compatible avec les commandes **FIXER LIMITE RECHERCHE** et **FIXER DESTINATION RECHERCHE**.

Rappel : Les références aux variables locales ne sont pas possibles en mode compilé. Pour plus d'informations sur la programmation SQL dans 4D, reportez-vous à la section **Présentation des commandes du thème SQL**.

A propos des liens

CHERCHER PAR SQL n'utilise pas les liens entre les tables définis dans l'éditeur de structure de 4D. Si vous souhaitez tirer parti des données liées, vous devez ajouter une clause JOIN dans la requête. Par exemple, considérons la structure suivante, dans laquelle un lien N vers 1 relie les champs [Personnes]Ville à [Villes]Nom :

```
[Personnes]
   Nom
   Ville
[Villes]
   Nom
   Population
```

Avec la commande **CHERCHER PAR FORMULE**, vous pourriez écrire :

```
CHERCHER PAR FORMULE([Personnes];[Villes]Population>1000)
```

Avec **CHERCHER PAR SQL**, vous devez écrire l'instruction suivante, que le lien existe ou non :

```
CHERCHER PAR SQL([Personnes];"personnes.ville=villes.nom AND villes.population>1000")
```

Note : Les liens 1 vers N et N vers N sont également traités par **CHERCHER PAR SQL** d'une manière différente de **CHERCHER PAR FORMULE**.

Exemple 1

Cet exemple recherche les bureaux dont les ventes sont supérieures à 100. La requête SQL est :

```
SELECT * FROM Bureaux WHERE Ventes > 100
```

En utilisant la commande **CHERCHER PAR SQL** :

```
C_ALPHA(30;$formuleRequete)
$formuleRequete:="Ventes > 100"
CHERCHER PAR SQL([Bureaux];$formuleRequete)
```

Exemple 2

Cet exemple recherche les commandes comprises entre 3000 et 4000. La requête SQL est :

```
SELECT *
FROM Commandes
WHERE Total BETWEEN 3000 AND 4000
```

En utilisant la commande **CHERCHER PAR SQL** :

```
C_ALPHA(40;$formuleRequete)
$formuleRequete:="Total BETWEEN 3000 AND 4000"
CHERCHER PAR SQL([Ventes];$formuleRequete)
```

Exemple 3

Cet exemple montre comment trier le résultat de la requête sur un critère spécifique. La requête SQL est :

```
SELECT *
FROM Personnes
WHERE Ville ='Paris'
ORDER BY Nom
```

En utilisant la commande **CHERCHER PAR SQL** :

```
C_ALPHA(40;$formuleRequete)
$formuleRequete:="Ville = 'Paris' ORDER BY Nom"
CHERCHER PAR SQL([Personnes];$formuleRequete)
```

Exemple 4

Cet exemple montre une requête utilisant des tables liées dans 4D. Via le SQL vous devez utiliser un JOIN pour recréer cette relation. Considérons les deux tables suivantes :

```
    [Factures] avec les champs (colonnes) suivants :
       ID_Fact : Entier long
       Date_Fact : Date
       Total : Réel
    [Lignes_Factures] avec les champs (colonnes) suivants :
       ID_Ligne : Entier long
       ID_Fact : Entier long
       Code : Alpha (10)
```

Un lien de N vers 1 relie le champ [Lignes_Factures]ID_Fact au champ [Factures]ID_Fact.
Avec la commande **CHERCHER PAR FORMULE**, vous pourriez écrire :

```
CHERCHER PAR FORMULE([Lignes_Factures];([Lignes_Factures]Code="FX-200") & (Mois de([Factures]Date_Fact)=4))
```

La requête SQL est :

```
SELECT ID_Ligne
FROM Lignes_Factures, Factures
WHERE Lignes_Factures.ID_Fact=Factures.ID_Fact
AND Lignes_Factures.Code='FX-200'
AND MONTH(Factures.Date_Fact) = 4
```

En utilisant la commande **CHERCHER PAR SQL** :

```
C_ALPHA(40;$formuleRequete)
$formuleRequete:="Lignes_Factures.ID_Fact=Factures.ID_Fact AND Lignes_Factures.Code='FX-200' AND
MONTH(Factures.Date_Fact)=4"
CHERCHER PAR SQL([Lignes_Factures];$formuleRequete)
```

Variables et ensembles système

Si le format de la condition de recherche est correct, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0, le résultat de la commande est une sélection vide et une erreur est retournée. Cette erreur peut être interceptée par une méthode installée à l'aide de la commande **APPELER SUR ERREUR**.

Debut SQL

Debut SQL
Ne requiert pas de paramètre

Description

Debut SQL est un mot-clé permettant d'indiquer dans l'éditeur de méthodes le début d'une séquence de commandes SQL, qui devront être interprétées par la source de données courante du process (moteur SQL intégré de 4D ou toute source définie via la commande **SQL LOGIN**).

Une séquence de commandes SQL initiée par **Debut SQL** doit être refermée par le mot-clé **Fin SQL**.

Les principes de fonctionnement de ces mots-clés sont les suivants :

- Vous pouvez placer un ou plusieurs blocs de balises **Debut SQL**/**Fin SQL** dans la même méthode. Vous pouvez générer des méthodes entièrement composées de code SQL ou mixer du code 4D et du code SQL dans la même méthode.
- Vous pouvez écrire plusieurs instructions SQL sur une même ligne ou sur différentes lignes en les séparant par un ";". Par exemple, vous pouvez écrire :

```
Debut SQL
   INSERT INTO SALESREPS (NAME, AGE) VALUES (Henry,40);
   INSERT INTO SALESREPS (NAME, AGE) VALUES (Bill,35)
Fin SQL
```

ou :

```
Debut SQL
   INSERT INTO SALESREPS (NAME, AGE) VALUES (Henry,40);INSERT INTO SALESREPS (NAME, AGE) VALUES (Bill,35)
Fin SQL
```

A noter que le **Débogueur** de 4D évaluera le code SQL ligne par ligne. Dans certains cas, il peut être préférable d'utiliser plusieurs lignes.

Fin SQL

Fin SQL
Ne requiert pas de paramètre

Description

Fin SQL est un mot-clé indiquant dans l'éditeur de méthodes la fin d'une séquence de commandes SQL.

Une séquence d'instructions SQL doit être encadrée par les mot-clés **Debut SQL** et **Fin SQL**. Pour plus d'informations, reportez-vous à la description du mot-clé **Debut SQL**.

FIXER VALEUR CHAMP NULL

FIXER VALEUR CHAMP NULL (leChamp)

Paramètre	Type		Description
leChamp	Champ	⇨	Champ auquel attribuer la valeur NULL

Description

La commande **FIXER VALEUR CHAMP NULL** attribue la valeur NULL au champ désigné par le paramètre *leChamp*.

La valeur NULL est exploitée par le moteur SQL de 4D. Pour plus d'informations, reportez-vous au manuel Guide de référence 4D SQL.

Notes :

- Il est possible d'interdire la valeur Null pour les champs 4D au niveau de l'éditeur de Structure (cf. manuel Mode Développement).
- **FIXER VALEUR CHAMP NULL** efface le contenu des champs objet.

LANCER SERVEUR SQL

LANCER SERVEUR SQL
Ne requiert pas de paramètre

Description

La commande **LANCER SERVEUR SQL** démarre le serveur SQL intégré de l'application 4D sur laquelle elle a été exécutée. Une fois lancé, le serveur SQL peut répondre aux requêtes SQL externes.

Note : Cette commande n'influe pas sur le fonctionnement du moteur SQL interne de 4D. Le moteur SQL est toujours disponible pour les requêtes internes.

Variables et ensembles système

Si le serveur SQL a été correctement lancé, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Lire source donnees courante

Lire source donnees courante -> Résultat

Paramètre	Type		Description
Résultat	Chaîne	➲	Nom de la source de données en cours d'utilisation

Description

La commande **Lire source donnees courante** retourne le nom de la source de données courante de l'application. La source de données courante reçoit les requêtes SQL exécutées au sein de structures **Debut SQL**/**Fin SQL**.
Lorsque la source de données courante est la base 4D locale, la commande retourne la chaîne ";DB4D_SQL_LOCAL;", correspondant à la valeur de la constante SQL_INTERNAL (thème "**SQL**").
Cette commande vous permet de contrôler la source de données courante, généralement avant d'exécuter une requête SQL.

LISTE SOURCES DONNEES

LISTE SOURCES DONNEES (typeSource ; tabNomsSources ; tabPilotes)

Paramètre	Type		Description
typeSource	Entier long	⇨	Type de source : utilisateur ou système
tabNomsSources	Tableau texte	⇦	Tableau des noms de sources de données
tabPilotes	Tableau texte	⇦	Tableau des pilotes des sources

Description

La commande **LISTE SOURCES DONNEES** retourne dans les tableaux *tabNomsSources* et *tabPilotes* les noms et les pilotes des sources de données de type *typeSource* définies dans le gestionnaire ODBC du système d'exploitation.

4D vous permet de vous connecter directement via le langage à une source de données ODBC externe et d'exécuter des requêtes SQL au sein d'une structure **Debut SQL**/**Fin SQL**. Le principe d'utilisation est le suivant : la commande **LISTE SOURCES DONNEES** permet d'obtenir la liste des sources de données présentes sur le poste. La commande **SQL LOGIN** permet alors de désigner la source à utiliser. Vous pouvez ensuite exécuter des requêtes SQL dans une structure **Debut SQL**/**Fin SQL** sur la source "courante". Pour effectuer à nouveau des requêtes sur le moteur interne de 4D, il suffit de passer la commande **SQL LOGOUT**. Pour plus d'informations sur les commandes SQL dans l'éditeur de méthodes, reportez-vous au manuel Guide de référence 4D SQL.

Passez dans *typeSource* le type de source de données que vous souhaitez obtenir. Vous pouvez utiliser l'une des constantes suivantes, placées dans le thème "**SQL**" :

Constante	Type	Valeur
Source de données système	Entier long	2
Source de données utilisateur	Entier long	1

Note : Cette commande ne prend pas en compte les sources de données de type fichier.

La commande remplit et dimensionne les tableaux *tabNomsSources* et *tabPilotes* avec les valeurs correspondantes.

Note : Si vous souhaitez vous connecter à une source de données 4D externe via ODBC, vous devez au préalable installer le pilote 4D ODBC sur votre poste. Pour plus d'informations, reportez-vous au manuel d'installation de 4D ODBC Driver.

Exemple

Cet exemple utilise une source de données utilisateur :

```
TABLEAU TEXTE(tdsn;0)
TABLEAU TEXTE(tdsnPilotes;0)
LISTE SOURCES DONNEES(Source de données utilisateur;tdsn;tdsnPilotes)
```

Variables et ensembles système

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 et une erreur est générée.

SQL ANNULER CHARGEMENT

SQL ANNULER CHARGEMENT
Ne requiert pas de paramètre

Description

La commande **SQL ANNULER CHARGEMENT** met fin à la requête SELECT courante et initialise les paramètres du curseur.

Cette commande permet d'exécuter plusieurs requêtes SELECT au sein d'une même connexion (c'est-à-dire un même curseur) initiée par la commande **SQL LOGIN**.

Exemple

Dans cet exemple, deux requêtes sont exécutées dans la même connexion :

```
C_BLOB(Monblob)
C_TEXTE(MonTexte)
SQL LOGIN("mysql";"root";"")

SQLStmt:="SELECT blob_field FROM app_testTable"
SQL EXECUTER(SQLStmt;Monblob)
Tant que(Non(SQL Fin de selection))
   SQL CHARGER ENREGISTREMENT
Fin tant que

 `Réinitialisation du curseur
SQL ANNULER CHARGEMENT

SQLStmt:="SELECT Name FROM Employee"
SQL EXECUTER(SQLStmt;MonTexte)
Tant que(Non(SQL Fin de selection))
   SQL CHARGER ENREGISTREMENT
Fin tant que
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL CHARGER ENREGISTREMENT

SQL CHARGER ENREGISTREMENT {(nombreEnr)}

Paramètre	Type		Description
nombreEnr	Entier long	⇨	Nombre d'enregistrements à charger

Description

La commande **SQL CHARGER ENREGISTREMENT** récupère dans 4D un ou plusieurs enregistrement(s) provenant de la source de données ouverte dans la connexion courante.

Le paramètre facultatif *nombreEnr* permet de définir le nombre d'enregistrements à récupérer :

- Si vous omettez ce paramètre, la commande récupèrera l'enregistrement courant dans la source de données. Ce principe correspond à la récupération des données dans une boucle où un enregistrement est reçu à la fois.
- Si vous passez une valeur entière dans *nombreEnr*, la commande récupérera *nombreEnr* enregistrements.
- Si vous passez la constante SQL tous les enregistrements (ou la valeur -1), la commande récupèrera tous les enregistrements de la table.

Note : Ces deux derniers paramétrages n'ont de sens que si les données récupérées sont associées à des tableaux ou des champs 4D.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL EXECUTER

SQL EXECUTER (instructionSQL {; objetLié}{; objetLié2 ; ... ; objetLiéN})

Paramètre	Type		Description
instructionSQL	Texte	⇒	Commande SQL à exécuter
objetLié	Variable, Champ	⇐	Réception du résultat (si nécessaire)

Description

La commande **SQL EXECUTER** permet d'exécuter une commande SQL et d'associer le résultat à des objets 4D (tableaux, variables ou champs) liés.

Pour que la commande puisse être exécutée, une connexion valide doit être définie dans le process courant.

Le paramètre *instructionSQL* contient la commande SQL à exécuter. Le paramètre *objetLié* reçoit les résultats. Les objets sont liés dans l'ordre de la colonne, ce qui signifie que les éventuelles colonnes distantes supplémentaires sont ignorées.

Si des champs 4D sont passés dans le(s) paramètre(s) *objetLié*, la commande créera des enregistrements et les sauvegardera automatiquement. Les champs doivent appartenir à la même table (il n'est pas possible de passer un champ de la table 1 et un champ de la table 2 dans le même appel). Si des champs de tables différentes sont passés, une erreur est générée.

Attention : Lorsque vous passez des champs 4D dans le(s) paramètre(s) *objetLié* et exécutez la commande **SELECT**, ce sont toujours les données de la source 4D distante qui sont modifiées. Si vous souhaitez récupérer en local des données de la source distante, vous devez utiliser des tableaux locaux intermédiaires et appeler la commande **INSERT** (cf. exemple 6).

Si vous passez des tableaux ou des variables 4D dans le(s) paramètre(s) *objetLié*, il est conseillé de les déclarer préalablement à l'appel de la commande afin de contrôler le type de données traitées. Les tableaux sont redimensionnés automatiquement si nécessaire.

Dans le cas d'une variable 4D, un seul enregistrement est récupéré à la fois.

Note : Pour plus d'informations sur le référencement des expressions 4D dans les requêtes SQL, reportez-vous à la section **Présentation des commandes du thème SQL**.

Exemple 1

Dans cet exemple, nous récupérons la colonne ename de la table emp dans la source de données. Le résultat est stocké dans le champ 4D [Employés]Nom. Les enregistrements 4D seront créés automatiquement :

```
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTER(SQLStmt;[Employés]Nom)
SQL CHARGER ENREGISTREMENT(SQL tous les enregistrements)
```

Exemple 2

Pour mieux contrôler la création des enregistrements, il est possible d'inclure le code au sein d'une transaction et de ne la valider que si le déroulement de l'opération s'est avéré satisfaisant :

```
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT alpha_field FROM app_testTable"
DEBUT TRANSACTION
SQL EXECUTER(SQLStmt;[Table 2]Champ1)
Tant que(Non(SQL Fin de selection))
   SQL CHARGER ENREGISTREMENT
   ... `Placer ici le code de validation des données
Fin tant que
VALIDER TRANSACTION `Validation de la transaction
```

Exemple 3

Dans cet exemple, nous récupérons la colonne ename de la table emp dans la source de données. Le résultat est stocké dans le tableau *tNoms*. Nous récupérons les enregistrements 10 par 10.

```
TABLEAU ALPHA(30;tNoms;20)
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTER(SQLStmt;tNoms)
Tant que(Non(SQL Fin de selection))
   SQL CHARGER ENREGISTREMENT(10)
Fin tant que
```

Exemple 4

Dans cet exemple, nous récupérons les colonnes ename et job de la table emp pour un ID spécifique (clause WHERE) de la source de données. Le résultat est stocké dans les variables 4D *vNom* and *vJob*. Seul le premier enregistrement est récupéré.

```
SQLStmt:="SELECT ename, job FROM emp WHERE id = 3"
SQL EXECUTER(SQLStmt;vName;vJob)
SQL CHARGER ENREGISTREMENT
```

Exemple 5

Dans cet exemple, nous récupérons la colonne Champ_Blob de la table Test dans la source de données. Le résultat est stocké dans une variable BLOB dont la valeur est mise à jour à chaque chargement d'enregistrement.

```
C_BLOB(MonBlob)
SQL LOGIN
SQL EXECUTER("SELECT Champ_Blob FROM Test";MonBlob)
Tant que(Non(SQL Fin de selection))
 `On parcourt le résultat
   SQL CHARGER ENREGISTREMENT
 `La valeur de MonBlob est mise à jour à chaque appel
Fin tant que
```

Exemple 6

Vous souhaitez récupérer en local des données stockées sur une base 4D Server distante. Pour cela, vous devez passer par des tableaux intermédiaires :

```
  // Connexion à la base distante
SQL LOGIN("IP:192.168.18.15:19812";"user";"password";*)
Si(OK=1)
       //A partir de ce point les requêtes sont adressées à la base distante
   C_TEXTE($LastName_value) // variable 4D utilisée dans la chaine de recherche
   TABLEAU TEXTE($a_LastName;0) // Stockage temporaire des valeurs distantes de LastName
   TABLEAU TEXTE($a_FirstName;0) // Stockage temporaire des valeurs distantes de FirstName
   C_BOOLEEN($UseSQL) //Choix du moyen de stocker en local
       // les données de la base distante (démonstration uniquement)

    $LastName_value:="Smith"  // Initialisation de la variable 4D

       // Associer la variable 4D $LastName_value avec le premier "?" dans la requête SQL
   SQL FIXER PARAMETRE($LastName_value;SQL paramètre entrée)

       // Récupérer de la table PERSONS distante les valeurs des champs LastName et FirstName
       // où "LastName = Smith" et les stocker dans les tableaux $a_LastName et $a_FirstName
   SQL EXECUTER("SELECT LastName, FirstName FROM PERSONS WHERE LastName = ?";$a_LastName;$a_FirstName)
   Si(Non(SQL Fin de selection))  // si au moins un enregistrement est trouvé

      SQL CHARGER ENREGISTREMENT(SQL tous les enregistrements)  // Charger tous les enregistrements

      $UseSQL:=Vrai  // Pour choisir la manière d'intégrer les données (démonstration uniquement)

      Si($UseSQL)  // Utilisation de requêtes SQL
         SQL LOGOUT  // Déconnexion de la base distante
         SQL LOGIN(SQL_INTERNAL;"user";"password")  // Connexion à la base locale
             //A partir de ce point les requêtes sont adressées à la base locale
             // Sauvegarde des tableaux $a_LastName et $a_FirstName dans la table locale PERSONS
         SQL EXECUTER("INSERT INTO PERSONS(LastName, FirstName) VALUES (:$a_LastName, :$a_FirstName);")

      Sinon   // Utilisation de commandes 4D
         Boucle($i;1;Taille tableau($a_LastName))
            CREER ENREGISTREMENT([PERSONS])
            [PERSONS]LastName:=$a_LastName{$i}
            [PERSONS]FirstName:=$a_FirstName{$i}
            STOCKER ENREGISTREMENT([PERSONS])
         Fin de boucle
      Fin de si
   Fin de si
   SQL LOGOUT  // Fermeture de la connexion
Fin de si
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL EXECUTER SCRIPT

SQL EXECUTER SCRIPT (cheminScript ; actionErreur {; nomAttribut ; valAttribut} {; nomAttribut2 ; valAttribut2 ; ... ; nomAttributN ; valAttributN})

Paramètre	Type		Description
cheminScript	Texte	➡	Chemin d'accès complet du fichier contenant le script SQL à exécuter
actionErreur	Entier long	➡	Action à effectuer en cas d'erreur durant l'exécution du script
nomAttribut	Texte	➡	Nom d'attribut à utiliser
valAttribut	Texte	➡	Valeur de l'attribut

Description

La commande **SQL EXECUTER SCRIPT** vous permet d'exécuter une suite d'instructions SQL placées dans le fichier de script désigné par *cheminScript*. Cette commande ne peut être exécutée que sur un poste local (4D local ou procédure stockée sur 4D Server). Elle fonctionne avec la base courante (base interne ou base externe).

Note : Cette commande ne peut pas être utilisée avec une connexion externe ouverte directement ou via ODBC.

Passez dans le paramètre *cheminScript* le chemin d'accès complet du fichier texte contenant les instructions SQL à exécuter. Le chemin d'accès doit être exprimé à l'aide de syntaxe du système courant. Si vous passez une chaîne vide ("") dans *cheminScript*, une boîte de dialogue standard d'ouverture de documents s'affiche, permettant à l'utilisateur de sélectionner le fichier de script à exécuter.

Note : Les commandes **SQL EXPORTER BASE** et **SQL EXPORTER SELECTION** génèrent automatiquement ce fichier de script.

Le paramètre *actionErreur* vous permet de paramétrer le fonctionnement de la commande lorsqu'elle rencontre une erreur au cours de l'exécution du script. Vous pouvez passer l'une des trois constantes ci-dessous, placées dans le thème **SQL** :

Constante	Type	Valeur	Comment
SQL abandonner si erreur	Entier long	1	En cas d'erreur, 4D stoppe immédiatement l'exécution du script.
SQL confirmer si erreur	Entier long	2	En cas d'erreur, 4D affiche une boîte de dialogue détaillant l'erreur et permettant à l'utilisateur d'interrompre ou de poursuivre l'exécution du script.
SQL continuer si erreur	Entier long	3	En cas d'erreur, 4D l'ignore et poursuit l'exécution du script.

Les paramètres *nomAttribut* et *valAttribut* doivent être passés par paires. Ces paramètres sont destinés à permettre de définir des attributs spécifiques pour l'exécution du script. Dans la version actuelle de 4D, un seul attribut peut être passé dans *nomAttribut*, disponible via la constante suivante, placée dans le thème **SQL** :

Constante	Type	Valeur	Comment
SQL utiliser les droits d'accès	Chaîne	SQL_Use_Access_Rights	Permet de restreindre les droits d'accès à appliquer lors de l'exécution des commandes SQL du script. Lorsque vous utilisez cet attribut, vous devez passer 0 ou 1 dans *valAttribut* : • *valAttribut* = 1 : 4D utilise les droits d'accès de l'utilisateur 4D courant. • *valAttribut* = 0 (ou attribut non défini) : 4D ne restreint pas les accès, les droits du Super_Utilisateur sont utilisés.

Si le fichier d'enregistrement des requêtes de 4D est activé (via les sélecteurs 28 ou 45 de la commande **FIXER PARAMETRE BASE**), chaque commande SQL exécutée génèrera une entrée avec les informations suivantes :

- Type de commande SQL
- Nombre d'enregistrements affectés par la commande
- Durée d'exécution de la commande
- Pour chaque erreur rencontrée :
  - le code d'erreur
  - le texte de l'erreur s'il est disponible

Si le script est correctement exécuté (aucune erreur rencontrée), la variable système *OK* prend la valeur 1. En cas d'erreur, la variable système *OK* prend ou non la valeur 0 en fonction du paramètre *actionErreur* :

- Si *actionErreur* vaut SQL abandonner si erreur (valeur 1), *OK* prend la valeur 0.
- Si *actionErreur* vaut SQL confirmer si erreur (valeur 2), la variable *OK* prend la valeur 0 si l'utilisateur choisit de stopper l'opération et 1 s'il choisit de la poursuivre.
- Si *actionErreur* vaut SQL continuer si erreur (valeur 3), la variable *OK* vaut toujours 1.

Note : Si vous utilisez cette commande pour exécuter des actions consommatrices de mémoire telles que l'importation massive de données, vous pouvez envisager de faire appel à la nouvelle commande SQL **ALTER DATABASE** afin de désactiver temporairement des options SQL.

SQL EXPORTER BASE

SQL EXPORTER BASE (cheminDossier {; nbFichiers {; tailleLimiteFichiers {; tailleLimiteChamps}}})

Paramètre	Type		Description
cheminDossier	Texte	⇒	Chemin d'accès du dossier d'export ou "" pour afficher une boîte de dialogue de sélection de dossier
nbFichiers	Entier long	⇒	Nombre maximum de fichiers par dossier
tailleLimiteFichiers	Entier long	⇒	Valeur de limite de taille des fichiers d'exportation (en Ko)
tailleLimiteChamps	Entier long	⇒	Limite de taille au-dessous de laquelle le contenu d'un champ Texte, BLOB ou Image sera intégré au fichier principal (en octets)

Description

La commande **SQL EXPORTER BASE** exporte au format SQL tous les enregistrements de toutes les tables de la base. En SQL, cette opération d'exportation globale est appelée "Dump".

Note : Cette commande ne peut pas être utilisée avec une connexion externe ouverte directement ou via ODBC.

Pour chaque table, la commande génère un fichier texte contenant les instructions SQL nécessaires à l'importation des données dans une autre base. Ce fichier peut être utilisé directement par la commande **SQL EXECUTER SCRIPT** afin d'importer les données dans une autre base 4D.

Les fichiers d'export seront placés dans un dossier nommé "SQLExport" créé dans le dossier de destination désigné par le paramètre *cheminDossier*. A noter que si un dossier "SQLExport" existe déjà à l'emplacement défini, la commande le remplace sans qu'aucun message d'alerte n'apparaisse.
Si vous passez une chaîne vide dans ce paramètre, 4D affiche une boîte de dialogue standard permettant à l'utilisateur de désigner le dossier de destination. Par défaut, la boîte de dialogue affiche le dossier courant de l'utilisateur ayant ouvert la session ("Mes Documents" sous Windows ou "Documents" sous Mac OS).

Pour chaque table exportée, la commande effectue les actions suivantes :

- un sous-dossier du nom de la table est créé dans le dossier de destination.
- un fichier texte nommé "Export.sql" est créé dans le sous-dossier. Ce fichier est encodé en UTF-8 avec BOM (marque d'ordre des octets). Il contient des ordres SQL **INSERT** correspondant aux données exportées. Les valeurs des champs sont séparées par des caractères deux-points. Il peut y avoir moins de valeurs que de champs dans la table. Dans ce cas, les champs restants seront considérés NULL.
- si la table contient des champs BLOB, image ou texte (textes à stockage externe, c'est-à-dire stockés en-dehors des enregistrements), par défaut la commande crée un sous-dossier supplémentaire nommé "BLOBS" à côté du fichier "Export.sql" et y crée autant de sous-dossiers que nécessaire nommés "BlobsN". Ces sous-dossiers stockeront sous forme de fichiers séparés le contenu de tous les champs BLOB, image ou texte externe des enregistrements de la table. Les fichiers BLOB sont nommés "BlobNNNNN.BLOB", les fichiers texte sont nommés "TEXTNNNNN.TXT" (où NNNNN est un nombre unique généré par l'application). Les fichiers image sont nommés PICTNNNNN.ZZZZ (où NNNNN est un nombre unique généré par l'application et ZZZZ est l'extension). Lorsque c'est possible, les images sont exportées dans leur format natif d'origine avec l'extension correspondant au type d'image (.jpg, .png, etc.). Si l'export au format natif n'est pas possible, les images sont exportées dans le format 4D interne avec l'extension .4PCT.
  Ce fonctionnement par défaut peut être modulé en fonction de la taille des données contenues dans le champ à l'aide du paramètre optionnel *tailleLimiteChamps* (cf. ci-dessous).
  Note : Ce fonctionnement diffère lorsque vous exécutez **SQL EXPORTER BASE** depuis un 4D en mode distant. Dans ce contexte, les données à stockage externe sont automatiquement incluses dans le fichier "Export.sql".

Si vous passez le paramètre *nbFichiers*, la commande créera autant de sous-dossiers "BlobsN" que nécessaire afin que chacun d'eux ne contienne pas plus de *nbFichiers* fichiers BLOBs, images ou textes externes. Par défaut, si le paramètre *nbFichiers* est omis, la commande limite le nombre de fichiers à 200. Si vous passez 0, chaque sous-dossier contiendra au plus un seul fichier.

Le paramètre *tailleLimiteFichiers* vous permet de définir une limite de taille (en ko) pour chaque fichier "Export.sql" créé sur le disque. Lorsque la taille du fichier d'export en cours de création atteint la valeur définie dans *tailleLimiteFichiers*, 4D stoppe l'écriture des enregistrements, referme le fichier et en crée un nouveau nommé "ExportN.sql" (où N représente le numéro de séquence) à côté du précédent. A noter qu'il s'agit d'une limite théorique : la taille effective des fichiers "ExportN.sql" dépasse la valeur définie dans *tailleLimiteFichiers* car le fichier n'est refermé qu'à l'issue de l'écriture complète de l'enregistrement en cours d'exportation (le contenu des enregistrements n'est pas fractionné). La valeur minimale acceptée est de 100 et la valeur maximale (valeur par défaut) est de 100 000 (100 Mo).

Le paramètre optionnel *tailleLimiteChamps* vous permet de définir une taille pivot au-dessous de laquelle le contenu d'un champ BLOB, image ou texte externe sera intégré au fichier principal "Export.sql" et non sauvegardé en tant que fichier séparé. Ce paramètre a pour but d'optimiser l'opération d'export en limitant le nombre de sous-dossiers et de fichiers créés sur le disque.
Le paramètre doit être exprimé en octets. Par exemple, si vous passez 1000, tous les champs BLOB, image ou texte externe contenant des données d'une taille inférieure ou égale à 1000 octets seront intégrés au fichier d'export principal.
A noter que les données des champs binaires (BLOB et image) intégrées au fichier d'export sont écrites au format hexadécimal, sous la forme X'0f20' (notation hexadécimale SQL standard, cf. **val_littérale**). Ce format est automatiquement pris en charge par le moteur SQL de 4D.
Par défaut, si le paramètre *tailleLimiteChamps* est omis, les champs BLOB, image et texte externe sont toujours exportés sous forme de fichiers externes, quelle que soit leur taille.

Dans le fichier d'export, il peut y avoir moins de valeurs que de champs dans la table. Dans ce cas, les champs vides seront considérés comme NULL. Vous pouvez également passer la valeur NULL dans un champ.

Si l'export s'est déroulé correctement, la variable *OK* prend la valeur 1. Dans le cas contraire, elle prend la valeur 0.

Note : Cette commande ne prend pas en charge les champs de type Objet.

SQL EXPORTER SELECTION

SQL EXPORTER SELECTION (laTable ; cheminDossier {; nbFichiers {; tailleLimiteFichiers {; tailleLimiteChamps}}})

Paramètre	Type		Description
laTable	Table	⇒	Table de laquelle exporter la sélection
cheminDossier	Texte	⇒	Chemin d'accès du dossier d'export ou "" pour afficher une boîte de dialogue de sélection de dossier
nbFichiers	Entier long	⇒	Nombre maximum de fichiers par dossier
tailleLimiteFichiers	Entier long	⇒	Valeur de limite de taille des fichiers d'export (en Ko)
tailleLimiteChamps	Entier long	⇒	Limite de taille au-dessous de laquelle le contenu d'un champ Texte, BLOB ou Image sera intégré au fichier principal (en octets)

Description

La commande **SQL EXPORTER SELECTION** exporte au format SQL les enregistrements de la sélection courante de la table 4D désignée par le paramètre *laTable*.

Cette commande est quasiment identique à la commande **SQL EXPORTER BASE**. Le fichier généré peut être utilisé directement par la commande **SQL EXECUTER SCRIPT** afin d'importer les données dans une autre base 4D. La seule différence entre ces deux commandes est le fait que **SQL EXPORTER SELECTION** exporte uniquement la sélection courante de *laTable* alors que **SQL EXPORTER BASE** exporte la totalité des données de la base. De même, à la différence de **SQL EXPORTER BASE**, la commande **SQL EXPORTER SELECTION** ne fonctionne pas avec les bases SQL externes. Elle ne peut être utilisée qu'avec la base principale.

Reportez-vous à la description de la commande **SQL EXPORTER BASE** pour le détail du fonctionnement et des paramètres de ces commandes.

Si la sélection courante est vide, la commande ne fait rien. A noter que dans ce cas, le dossier de destination n'est pas vidé.

Si l'export s'est déroulé correctement, la variable *OK* prend la valeur 1. Dans le cas contraire, elle prend la valeur 0.

Note : Cette commande ne prend pas en charge les champs de type Objet.

SQL Fin de selection

SQL Fin de selection -> Résultat

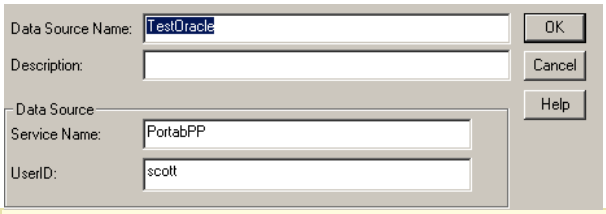
Paramètre	Type		Description
Résultat	Booléen	➲	Limites de l'ensemble de recherche atteintes

Description

La commande **SQL Fin de selection** indique si les limites de l'ensemble résultat ont été atteintes.

Exemple

Le code ci-dessous se connecte à une source de données externe (Oracle) à l'aide des paramètres suivants :




```
C_TEXTE(vName)

SQL LOGIN("TestOracle";"scott";"tiger")
Si(OK=1)
   SQL EXECUTER("SELECT ename FROM emp";vName)
   Tant que(Non(SQL Fin de selection))
      SQL CHARGER ENREGISTREMENT
   Fin tant que
   SQL LOGOUT
Fin de si
```

Cet exemple retournera dans la variable 4D *vName* les noms (ename) stockés dans la table nommée emp.

SQL FIXER OPTION

SQL FIXER OPTION (option ; valeur)

Paramètre	Type		Description
option	Entier long	⇒	Numéro d'option à définir
valeur	Entier long, Chaîne	⇒	Nouvelle valeur de l'option

Description

La commande **SQL FIXER OPTION** permet de modifier la *valeur* de l'option passée dans le paramètre *option*.

Vous pouvez passer dans *option* l'une des constantes suivantes, placées dans le thème **SQL** :

Constante	Type	Valeur	Comment
SQL asynchrone	Entier long	1	0 = connexion synchrone (valeur par défaut), 1 (ou valeur différente de 0) = connexion asynchrone
SQL jeu de caractères	Entier long	100	Encodage du texte utilisé pour les requêtes envoyées aux sources externes (via le SQL pass-through). La modification est effective pour le process courant et la connexion courante. Valeurs possibles : identifiant MIBEnum (cf. note 2) ou valeur -2 (cf. note 3) Par défaut : 106 (UTF-8)
SQL longueur maxi données	Entier long	3	Longueur maximale des données retournées
SQL nombre maxi lignes	Entier long	2	Nombre maximum de lignes dans l'ensemble résultant (utilisé pour les prévisualisations)
SQL timeout connexion	Entier long	5	Durée maximale d'attente lors de l'exécution de la commande SQL LOGIN. Cette valeur doit être fixée avant l'ouverture de la connexion pour être prise en compte Valeurs possibles : durée en secondes Par défaut : pas de timeout
SQL timeout requête	Entier long	4	Durée maximale d'attente de la réponse lors de l'exécution de la commande SQL EXECUTER. Valeurs : durée en secondes Par défaut : pas de timeout

Notes :

1. Lorsque vous travaillez avec le moteur SQL interne de 4D, l'option SQL asynchrone est inutile. En effet, ce type de connexion est toujours synchrone.
2. Les numéros MIBEnum sont référencés à l'adresse suivante : http://www.iana.org/assignments/character-sets.
3. Lorsque vous passez -2 comme *valeur* à SQL jeu de caractères, l'encodage utilisé par le serveur SQL de 4D est automatiquement adapté à la plate-forme d'exécution (encodage non-UTF) :

- sous Windows, ISO8859-1 est utilisé,
- sous Mac OS, MAC-ROMAN est utilisé.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL FIXER PARAMETRE

SQL FIXER PARAMETRE (objet ; typeParam)

Paramètre	Type		Description
objet	Objet 4D	➡	Objet 4D à utiliser (variable, tableau ou champ)
typeParam	Entier long	➡	Type du paramètre

Description

La commande **SQL FIXER PARAMETRE** permet d'utiliser la valeur d'une variable, d'un tableau ou d'un champ 4D dans les requêtes SQL.

Note : Il est également possible d'insérer directement le nom d'un objet 4D à utiliser (variable, tableau ou champ) entre les caractères << et >> dans le texte de la requête (cf. exemple 1). Pour plus d'informations sur ce point, reportez-vous à la section **Présentation des commandes du thème SQL**.

- Passez dans le paramètre *objet* l'objet 4D (variable, tableau ou champ) à utiliser dans la requête.
- Passez dans le paramètre *typeParam* le type SQL du paramètre. Vous pouvez passer une valeur ou utiliser l'une des constantes suivantes, placées dans le thème "**SQL**" :

Constante	Type	Valeur	Comment
SQL paramètre entrée	Entier long	1	
SQL paramètre entrée sortie	Entier long	2	Utilisable uniquement dans le contexte d'une procédure stockée SQL (paramètre entrée-sortie défini dans la procédure stockée)
SQL paramètre sortie	Entier long	4	Utilisable uniquement dans le contexte d'une procédure stockée SQL (paramètre sortie défini dans la procédure stockée)

La valeur de l'objet 4D est substituée au caractère *?* dans la requête SQL (syntaxe standard). Si la requête comporte plusieurs caractères *?*, plusieurs appels à **SQL FIXER PARAMETRE** seront nécessaires. Les valeurs des objets 4D seront affectées séquentiellement dans la requête, dans l'ordre d'exécution des commandes.

Attention : Cette commande permet de manipuler les *paramètres* passés à la requête SQL. Il n'est pas possible d'utiliser le type SQL paramètre sortie pour associer un objet 4D au *résultat* d'une requête SQL. Le résultat d'une requête SQL est récupéré par exemple via le paramètre *objetLié* de la commande **SQL EXECUTER** (cf. section **Présentation des commandes du thème SQL**). La commande **SQL FIXER PARAMETRE** est généralement utilisée pour définir des paramètres passés à la requête (SQL paramètre entrée) ; les types SQL paramètre sortie et SQL paramètre entrée sortie sont réservés à une utilisation dans le contexte de procédures stockées SQL pouvant retourner des paramètres.

Exemple 1

Cet exemple permet d'exécuter une requête SQL faisant directement appel à des variables 4D associées :

```
C_TEXTE(MonTexte)
C_ENTIER LONG(MonEntierLong)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<MonTexte>>, <<MonEntierLong>>)"
Boucle(vCounter;1;10)
   MonTexte:="Texte"+Chaine(vCounter)
   MonEntierLong:=vCounter
   SQL EXECUTER(SQLStmt)
   SQL ANNULER CHARGEMENT
Fin de boucle
SQL LOGOUT
```

Exemple 2

Même exemple que le précédent, mais en utilisant la commande **SQL FIXER PARAMETRE** :

```
C_TEXTE(MonTexte)
C_ENTIER LONG(MonEntierLong)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (?,?)"
Boucle(vCounter;1;10)
   MonTexte:="Texte"+Chaine(vCounter)
   MonEntierLong:=vCounter
   SQL FIXER PARAMETRE(MonTexte;SQL paramètre entrée)
   SQL FIXER PARAMETRE(MonEntierLong;SQL paramètre entrée)
   SQL EXECUTER(SQLStmt)
   SQL ANNULER CHARGEMENT
Fin de boucle
SQL LOGOUT
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL LIRE DERNIERE ERREUR

SQL LIRE DERNIERE ERREUR (errCode ; errTexte ; errODBC ; errSQLServer)

Paramètre	Type		Description
errCode	Entier long	⇦	Code de l'erreur
errTexte	Texte	⇦	Texte de l'erreur
errODBC	Texte	⇦	Code d'erreur ODBC
errSQLServer	Entier long	⇦	Code d'erreur native serveur SQL

Description

La commande **SQL LIRE DERNIERE ERREUR** retourne des informations relatives à la dernière erreur rencontrée lors de l'exécution d'une commande ODBC. L'erreur peut provenir de l'application 4D, du réseau, de la source ODBC, etc.

Cette commande doit généralement être appelée dans le contexte d'une méthode de gestion des erreurs installée à l'aide de la commande **APPELER SUR ERREUR**.

- Le paramètre *errCode* retourne le code de l'erreur.
- Le paramètre *errTexte* retourne le libellé de l'erreur.

Les deux derniers paramètres ne sont remplis que si l'erreur provient de la source ODBC. Dans le cas contraire, ils sont retournés vides.

- Le paramètre *errODBC* retourne le code d'erreur ODBC (SQL state).
- Le paramètre *errSQLServer* retourne le code de l'erreur native du serveur SQL.

SQL LIRE OPTION

SQL LIRE OPTION (option ; valeur)

Paramètre	Type		Description
option	Entier long	⇨	Numéro d'option
valeur	Entier long, Texte	⇦	Valeur de l'option

Description

La commande **SQL LIRE OPTION** retourne la *valeur* courante de l'option passée dans le paramètre *option*.
Pour plus d'informations sur les différentes options et leurs valeurs associées, reportez-vous à la description de la commande **SQL FIXER OPTION**.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL LOGIN

SQL LOGIN {(source ; nomUtilisateur ; motDePasse ; *)}

Paramètre	Type		Description
source	Chaîne	⇒	Nom de publication de base 4D ou Adresse IP de base distante ou Nom de source de données dans le gestionnaire ODBC ou "" pour afficher le dialogue de sélection
nomUtilisateur	Chaîne	⇒	Nom d'utilisateur enregistré dans la source de données
motDePasse	Chaîne	⇒	Mot de passe de l'utilisateur
*	Opérateur	⇒	Appliquer à Debut SQL/Fin SQL Si omis : ne pas appliquer (base locale), si passé : appliquer

Description

La commande **SQL LOGIN** vous permet d'ouvrir une connexion avec une source de données SQL, définie dans le paramètre *source*. Elle désigne la cible des requêtes SQL exécutées ultérieurement dans le process courant :

- via la commande **SQL EXECUTER**,
- via le code placé à l'intérieur des balises **Debut SQL/Fin SQL** (si le paramètre * est passé).

La source de données SQL peut être soit :

- une base 4D Server externe à laquelle vous accédez directement,
- une source ODBC externe,
- la base 4D locale (base interne).

Vous pouvez passer dans *source* l'une des valeurs suivantes : une adresse IP, un nom de publication de base 4D, un nom de source de données ODBC, une chaîne vide ou la constante SQL_INTERNAL.

- **adresse IP**
  Syntaxe : **IP:<Adresse IP>{:<PortTCP>}**
  Dans ce cas, la commande ouvre une connexion directe avec la base 4D Server exécutée sur l'ordinateur ayant l'adresse IP définie. Sur l'ordinateur "cible", le serveur SQL doit être lancé. Si vous passez un numéro de port TCP, il doit avoir été spécifié comme port de publication du serveur SQL dans la base "cible". Si vous ne passez pas de numéro de port TCP, le port par défaut sera utilisé (19812). Le numéro de port TCP du serveur SQL peut être modifié dans la page "SQL" des Propriétés de la base. Reportez-vous aux exemples 4 et 5.
  Si vous avez activé le SSL pour le serveur SQL "cible" (option accessible via les Propriétés de la base), vous devez ajouter le mot-clé ":ssl" à la suite de l'adresse IP et du port TCP (obligatoire dans ce cas) afin que le serveur puisse traiter correctement la requête (voir exemple 6).

- **nom de publication de base 4D**
  Syntaxe : **4D:<Nom_de_Publication>**
  Dans ce cas, la commande ouvre une connexion directe avec la base 4D Server dont le nom de publication sur le réseau correspond au nom spécifié. Le nom de publication réseau d'une base est défini dans la page "Client-Serveur" des Propriétés de la base.
  Reportez-vous à l'exemple 4.
  Note : Le numéro de port TCP du serveur SQL 4D cible (qui publie la base 4D) et le numéro de port TCP du serveur SQL de l'application 4D ouvrant la connexion doivent être identiques.

- **nom de source de données ODBC valide**
  Syntaxe : **ODBC:<Ma_DSN>** ou **<Ma_DSN>**
  Dans ce cas, le paramètre *source* contient le nom de la source de données telle qu'elle a été définie dans le gestionnaire du pilote ODBC.
  Notes :
  - Par compatibilité avec les versions précédentes de 4D, il est possible d'omettre le préfixe "ODBC:". Toutefois pour des raisons de lisibilité du code il est conseillé d'utiliser ce préfixe. Reportez-vous à l'exemple 2.
  - Sous Windows, le nom de la source de données doit respecter les majuscules/minuscules. Par exemple, si la source de données a été définie en "4D_v16", passer la valeur "4D_V16" échouera.
  - Sous Windows et Mac, le préfixe "ODBC:" doit être saisi en majuscules. Si vous passez "odbc:", la connexion échouera.

- **chaîne vide**
  Syntaxe : **""**
  Dans ce cas la commande provoque l'affichage de la boîte de dialogue de connexion, permettant de désigner manuellement la source de données à laquelle se connecter :

  Cette boîte de dialogue comporte plusieurs pages. La page TCP/IP se compose des éléments suivants :
  - Nom cible : ce menu est construit à l'aide de deux listes :
    - la liste des bases ouvertes récemment en connexion directe. Le mécanisme de mise à jour de cette liste est identique à celui de l'application 4D, à la différence près que le dossier contenant les fichiers .4DLink est nommé "Favorites SQL vXX" au lieu de "Favorites vXX".
    - la liste des applications 4D Server dont le serveur SQL est lancé et dont le port TCP pour les connexions SQL est égal à celui de l'application source. Cette liste est mise à jour dynamiquement à chaque nouvel appel de la commande **SQL LOGIN** sans le paramètre *source*. Le caractère "^" placé devant un nom de base indique que la connexion est effectuée en mode sécurisé via SSL.
  - Adresse réseau : cette zone affiche l'adresse IP et éventuellement le port TCP de la base sélectionnée dans le menu Nom cible. Vous pouvez également saisir dans cette zone une adresse IP puis cliquer sur le bouton Connexion afin de vous connecter à la base 4D Server correspondante. Vous pouvez également spécifier le port TCP, en saisissant deux points (:) puis le numéro du port à la suite de l'adresse. Par exemple : 192.168.93.105:19855
  - Utilisateur et Mot de passe : ces zones permettent de saisir les identifiants de la connexion.
  - Les pages DSN utilisateur et DSN système affichent respectivement la liste des sources de données ODBC utilisateur et système définies dans le gestionnaire ODBC de la machine. Ces pages permettent de sélectionner une source de données et de saisir des identifiants afin d'ouvrir une connexion avec une source ODBC externe.

  Si la connexion est établie, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 et une erreur est générée. Cette erreur peut être interceptée via une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

- **constante SQL_INTERNAL**
  Syntaxe : SQL_INTERNAL
  Dans ce cas, la commande redirige les requêtes SQL suivantes vers la base 4D interne.

Attention : Les préfixes utilisés dans le paramètre *source* (IP, ODBC, 4D) doivent être écrits en majuscules.

Le paramètre *utilisateur* contient le nom de l'utilisateur autorisé à se connecter à la source de données externe. Par exemple, avec Oracle®, ce nom d'utilisateur peut être "Scott".

Le paramètre *motDePasse* contient le mot de passe de l'utilisateur autorisé à se connecter. Par exemple, avec Oracle®, ce mot de passe peut être "tiger".

Note : Dans le cas d'une connexion directe, si vous passez des chaînes vides dans les paramètres *utilisateur* et *motDePasse*, la connexion ne sera acceptée que si les mots de passe 4D ne sont pas activés dans la base cible. Sinon, la connexion est refusée.

Le paramètre facultatif * permet de changer la cible du code SQL exécuté au sein des balises **Debut SQL**/**Fin SQL**. Si vous ne passez pas ce paramètre, le code placé dans les balises **Debut SQL**/**Fin SQL** sera toujours adressé au moteur SQL interne de 4D, sans tenir compte du paramétrage défini par la commande **SQL LOGIN**. Si vous passez ce paramètre, le code SQL exécuté au sein des balises **Debut SQL**/**Fin SQL** sera adressé à la *source* définie par la commande.

Pour refermer la connexion courante et libérer la mémoire, il suffit d'exécuter la commande **SQL LOGOUT**. Toutes les requêtes SQL sont alors dirigées vers la base 4D SQL interne.
Si vous appelez une nouvelle fois **SQL LOGIN** sans avoir refermé explicitement la connexion courante, elle est automatiquement refermée.

Note : En cas d'échec d'une tentative de connexion externe via **SQL LOGIN**, la base 4D interne devient automatiquement la source de données courante.

Tous les paramètres sont facultatifs. Si aucun paramètre n'est passé, la commande provoquera l'affichage de la boîte de dialogue de connexion ODBC, permettant de désigner manuellement la source de données à laquelle se connecter.

La portée de cette commande est le process. Autrement dit, si vous souhaitez ouvrir deux connexions distinctes, vous devez créer deux process et ouvrir chaque connexion dans chaque process.

Attention : Il n'est pas possible d'ouvrir une connexion ODBC dans les contextes décrits ci-dessous. Ces configurations conduisent au blocage de l'application :

- connexion via ODBC depuis l'application en exécution vers elle-même
- connexion via ODBC depuis une application 4D vers 4D Server alors qu'une connexion client/serveur classique est déjà ouverte entre les deux applications.

Exemple 1

Cette instruction provoque l'affichage de la boîte de dialogue du gestionnaire ODBC :

```
SQL LOGIN
```

Exemple 2

Ouverture d'une connexion via le protocole ODBC avec la source de données externe "MonOracle". Les requêtes SQL exécutées via la commande **SQL EXECUTER** et les requêtes incluses dans les balises **Debut SQL**/**Fin SQL** seront redirigées vers cette connexion :

```
SQL LOGIN("ODBC:MonOracle";"Scott";"tiger";*)
```

Exemple 3

Ouverture d'une connexion avec le moteur SQL interne de 4D :

```
SQL LOGIN(SQL_INTERNAL;$utilisateur;$motdepasse)
```

Exemple 4

Ouverture d'une connexion directe avec l'application 4D Server exécutée sur le poste ayant l'adresse IP 192.168.45.34 et répondant sur le port TCP par défaut. Les requêtes SQL exécutées via la commande **SQL EXECUTER** seront redirigées vers cette connexion, les requêtes incluses dans les balises **Debut SQL**/**Fin SQL** ne seront pas redirigées.

```
SQL LOGIN("IP:192.168.45.34";"John";"azerty")
```

Exemple 5

Ouverture d'une connexion directe avec l'application 4D Server exécutée sur le poste ayant l'adresse IP 192.168.45.34 et répondant sur le port TCP 20150. Les requêtes SQL exécutées via la commande **SQL EXECUTER** et les requêtes incluses dans les balises **Debut SQL**/**Fin SQL** seront redirigées vers cette connexion.

```
SQL LOGIN("IP:192.168.45.34:20150";"John";"azerty";*)
```

Exemple 6

Ouverture d'une connexion directe en SSL avec l'application 4D Server exécutée sur le poste ayant l'adresse IP 192.168.45.34 et répondant sur le port TCP par défaut. Le SSL doit avoir été activé pour le serveur SQL sur l'application 4D Server :

```
SQL LOGIN("IP:192.168.45.34:19812:ssl";"Admin";"sd156") // Notez le ":ssl" après l'adresse IP et le port TCP
```

Exemple 7

Ouverture d'une connexion directe avec l'application 4D Server qui publie sur le réseau local une base dont le nom de publication est "DB_Compta". Le port TCP utilisé pour le serveur SQL des deux bases (défini dans la page "SQL" des Propriétés de la base) doit être identique (19812 par défaut). Les requêtes SQL exécutées via la commande **SQL EXECUTER** seront redirigées vers cette connexion, les requêtes incluses dans les balises **Debut SQL**/**Fin SQL** ne seront pas redirigées.

```
SQL LOGIN("4D:DB_Compta";"John";"azerty")
```

Exemple 8

Cet exemple illustre les possibilités de connexion offertes par la commande **SQL LOGIN** :

```
TABLEAU TEXTE(aNames;0)
TABLEAU ENTIER LONG(aAges;0)
SQL LOGIN("ODBC:MonORACLE";"Marc";"azerty")
Si(OK=1)
  `La requête suivante sera redirigée vers la base ORACLE externe
   SQL EXECUTER("SELECT Name, Age FROM PERSONS";aNames;aAges)
  `La requête suivante sera dirigée vers la base 4D locale
   Debut SQL
      SELECT Name, Age
      FROM PERSONS
      INTO :aNames, :aAges;
   Fin SQL
  `La commande SQL LOGIN suivante referme la connexion courante
  `avec la base externe ORACLE et ouvre une nouvelle connexion avec
  `une base externe MySQL
   SQL LOGIN("ODBC:MySQL";"Jean";"qwerty";*)
   Si(OK=1)
  `La requête suivante sera redirigée vers la base MySQL externe
      SQL EXECUTE("SELECT Name, Age FROM PERSONS";aNames;aAges)
  `La requête suivante sera aussi redirigée vers la base MySQL externe
      Debut SQL
         SELECT Name, Age
         FROM PERSONS
         INTO :aNames, :aAges;
      Fin SQL
      SQL LOGOUT
  `La requête suivante sera dirigée vers la base 4D locale
      Debut SQL
         SELECT Name, Age
         FROM PERSONS
         INTO :aNames, :aAges;
      Fin SQL
   Fin de si
Fin de si
```

Variables et ensembles système

Si la connexion est correctement établie, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

SQL LOGOUT

SQL LOGOUT
Ne requiert pas de paramètre

Description

La commande **SQL LOGOUT** referme la connexion avec une source ODBC ouverte dans le process courant (le cas échéant). S'il n'y a pas de connexion ODBC ouverte, la commande ne fait rien.

Variables et ensembles système

Si la connexion a été correctement refermée, la variable système OK prend la valeur 1, sinon elle prend la valeur 0. Vous pouvez intercepter les éventuelles erreurs à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

Valeur champ Null

Valeur champ Null (leChamp) -> Résultat

Paramètre	Type		Description
leChamp	Champ	⇒	Champ à évaluer
Résultat	Booléen	⇐	Vrai = le champ est NULL, Faux = le champ n'est pas NULL

Description

La commande **Valeur champ Null** retourne **Vrai** si le champ désigné par le paramètre *leChamp* contient la valeur NULL, et **Faux** sinon.

La valeur NULL est exploitée par le moteur SQL de 4D. Pour plus d'informations, reportez-vous au Manuel de référence SQL 4D.

La valeur retournée par cette commande n'a de sens que si l'option "" n'est pas cochée dans la définition du champ en Structure. Dans le cas contraire, elle retourne toujours **Faux**.

_o_UTILISER BASE EXTERNE

_o_UTILISER BASE EXTERNE (nomSource {; nomUtilisateur ; motDePasse})

Paramètre	Type		Description
nomSource	Chaîne	⇒	Nom de la source de données ODBC à laquelle se connecter
nomUtilisateur	Chaîne	⇒	Nom d'utilisateur
motDePasse	Chaîne	⇒	Mot de passe de l'utilisateur

Note de compatibilité

Cette commande est remplacée par la commande **SQL LOGIN** depuis la version 11.3 de 4D. Elle ne doit plus être utilisée dans vos développements.

_o_UTILISER BASE INTERNE

_o_UTILISER BASE INTERNE
Ne requiert pas de paramètre

Note de compatibilité

Cette commande est remplacée par la commande **SQL LOGOUT** depuis la version 11.3 de 4D. Elle ne doit plus être utilisée dans vos développements.

SVG

- Présentation des commandes SVG
- SVG Chercher ID element par coordonnees
- SVG Chercher ID elements par rect
- SVG EXPORTER VERS IMAGE
- SVG FIXER ATTRIBUT
- SVG LIRE ATTRIBUT
- SVG MONTRER ELEMENT

Présentation des commandes SVG

SVG (Scalable Vector Graphics) est un format de fichier permettant de décrire en XML un graphique vectoriel (extension .svg). L'utilisation la plus courante du SVG est la publication de données statistiques ou cartographiques.

Ces fichiers peuvent être visualisés dans les navigateurs Web, soit nativement, soit via des plug-ins. 4D comporte un moteur de rendu SVG vous permettant de visualiser les fichiers SVG dans les champs ou les variables image. La commande **SVG EXPORTER VERS IMAGE** vous permet de générer une image dans 4D à partir d'une description SVG. A noter également que la commande **GRAPHE** permet de tirer parti du moteur SVG intégré de 4D.

Pour plus d'informations sur ce format, reportez-vous à l'adresse http://www.w3.org/Graphics/SVG/.

SVG Chercher ID element par coordonnees

SVG Chercher ID element par coordonnees ({* ;} objetImage ; x ; y) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objetImage est un nom d'objet (chaîne) Si omis, objetImage est un champ ou une variable
objetImage	Image	⇒	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
x	Entier long	⇒	Coordonnée X en pixels
y	Entier long	⇒	Coordonnée Y en pixels
Résultat	Chaîne	⇐	ID de l'élément se trouvant à l'emplacement x,y

Description

La commande **SVG Chercher ID element par coordonnees** retourne l'ID (attribut "id" ou "xml:id") de l'élément XML situé à l'emplacement défini par les coordonnées (*x*,*y*) dans l'image SVG désignée par le paramètre *objetImage*. Cette commande permet notamment de créer des interfaces graphiques interactives utilisant des objets SVG.

Note : Pour plus d'informations sur le format SVG, reportez-vous à la section **Présentation des commandes XML génériques**.

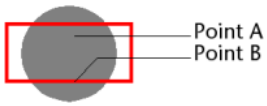
Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objetImage* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objetImage* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

A noter qu'il n'est pas obligatoire que l'image soit affichée dans un formulaire. Dans ce cas, la syntaxe de type "nom d'objet" n'est pas valide, vous devez passer un nom de champ ou de variable.

Les coordonnées passées dans les paramètres *x* et *y* doivent être exprimées en pixels relativement à l'angle supérieur gauche de l'image (0,0). Dans le contexte d'une image affichée dans un formulaire, vous pouvez utiliser les valeurs retournées par les *variables système* MouseX et MouseY. Ces variables sont mises à jour dans les événements formulaire Sur clic, Sur double clic et Sur relâchement bouton ainsi que Sur début survol et Sur survol.

Note : Dans le système de coordonnées des images, MouseX et MouseY définissent toujours le même point de l'image, quel que soit son format d'affichage (hormis pour le format "mosaïque"), même si l'image a défilé ou a été zoomée.

Le point pris en compte est le premier point atteint. Par exemple, dans le cas ci-dessous, la commande retournera l'ID du cercle si les coordonnées du point A sont passées et celui du rectangle si les coordonnées du point B sont passées :



Si les coordonnées correspondent à des objets superposés ou composites, la commande retourne l'ID du premier objet disposant d'un attribut ID valide en remontant si nécessaire parmi les éléments parents.

La commande retourne une chaîne vide si :

- la racine est atteinte sans qu'un attribut "id" ait été trouvé,
- le point de coordonnées n'appartient à aucun objet,
- l'attribut "id" est une chaîne vide.

Note : Cette commande ne permet pas de détecter des objets dont la valeur d'opacité (attribut "fill-opacity") est inférieure à 0,01

Variables et ensembles système

Si *objetImage* ne contient pas une image SVG valide, la commande retourne une chaîne vide et la variable système OK prend la valeur 0. Sinon, si la commande a été exécutée correctement, la variable système OK prend la valeur 1.

SVG Chercher ID elements par rect

SVG Chercher ID elements par rect ({* ;} objetImage ; x ; y ; largeur ; hauteur ; tabIds) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objetImage est un nom d'objet (chaîne) Si omis, objetImage est un champ ou une variable
objetImage	Image	⇒	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
x	Entier long	⇒	Coordonnée horizontale du coin haut gauche du rectangle de sélection
y	Entier long	⇒	Coordonnée verticale du coin haut gauche du rectangle de sélection
largeur	Entier long	⇒	Largeur du rectangle de sélection
hauteur	Entier long	⇒	Hauteur du rectangle de sélection
tabIds	Tableau texte	⇐	IDs des éléments dont le rectangle englobant est en intersection avec le rectangle de sélection
Résultat	Booléen	⇐	Vrai = au moins un élément est trouvé, Faux sinon

Description

La commande **SVG Chercher ID elements par rect** remplit le tableau texte *tabIds* avec les IDs (attribut "id" ou "xml:id") des éléments XML dont le rectangle englobant est en intersection avec le rectangle de sélection à l'emplacement défini par les paramètres *x* et *y*.

La commande retourne Vrai si au moins un élément est trouvé (c'est-à-dire si le tableau *tabIds* est non vide) et Faux sinon.

Cette commande permet notamment de gérer des interfaces graphiques interactives.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objetImage* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objetImage* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).
Si vous travaillez avec un champ ou une variable image, la commande utilise l'image d'origine, correspondant à la source de données (*datasource*). En revanche, si vous travaillez avec un objet du formulaire, la commande utilise l'image courante, éventuellement modifiée via la commande **SVG FIXER ATTRIBUT** et qui est conservée avec les propriétés de l'objet du formulaire.

Les coordonnées passées dans les paramètres *x* et *y* doivent être exprimées en pixels relativement à l'angle supérieur gauche de l'image (0,0). Vous pouvez utiliser les valeurs retournées par les **Variables système** MouseX et MouseY. Ces variables sont mises à jour dans les événements formulaire Sur clic, Sur double clic ainsi que Sur début survol et Sur survol.

Note : Dans le système de coordonnées des images, [x;y] définit toujours le même point, quel que soit le format d'affichage de l'image, hormis pour le format "mosaïque".

Tous les ID d'éléments dont le rectangle englobant est en intersection avec le rectangle de sélection sont pris en compte, même ceux situés sous d'autres éléments.

SVG EXPORTER VERS IMAGE

SVG EXPORTER VERS IMAGE (refElément ; vVarImage {; typeExport})

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML racine
vVarImage	Image	➡	Variable image devant recevoir l'arbre XML (image SVG)
typeExport	Entier long	➡	0=Ne pas stocker la source de données, 1=Copier la source de données, 2 = Prendre possession de la source de données (défaut)

Description

La commande **SVG EXPORTER VERS IMAGE** permet de sauvegarder dans la variable ou le champ image désigné(e) par le paramètre *vVarImage* une image au format SVG contenue dans un arbre XML.

Note : Pour plus d'informations sur le format SVG, reportez-vous à la section **Présentation des commandes XML génériques**.

Passez dans *refElément* la référence de l'élément XML racine contenant l'image SVG.

Passez dans *vVarImage* le nom de la variable image ou du champ image 4D devant contenir l'image SVG. L'image est exportée dans son format natif (description XML) et est dessinée via le moteur de rendu SVG au moment de l'affichage.

Le paramètre facultatif *typeExport* vous permet de définir la manière dont la source de données XML doit être prise en charge par la commande. Vous pouvez passer dans ce paramètre une des constantes suivantes, placées dans le thème "**XML**" :

Constante	Type	Valeur	Comment
Copier source données XML	Entier long	1	4D conserve une copie de l'arbre DOM avec l'image, ce qui permet de la sauvegarder dans un champ image de la base de données et de la réafficher ou de l'exporter à tout moment.
Lire source données XML	Entier long	0	4D lit uniquement la source de données XML, elle n'est pas conservée avec l'image. Ce paramétrage accélère sensiblement l'exécution de la commande, toutefois l'arbre DOM n'étant pas conservé, il ne sera pas possible de stocker ni d'exporter l'image.
Posséder source données XML	Entier long	2	4D exporte l'arbre DOM avec l'image. L'image pourra être stockée ou exportée et l'exécution de la commande est rapide. Toutefois, la référence XML *refElément* n'est alors plus utilisable par les autres commandes 4D. Ce mode d'exportation est utilisé par défaut si le paramètre *typeExport* est omis.

Exemple

L'exemple suivant permet d'afficher "Hello World" dans une image 4D :

```
C_IMAGE(vImage)
$svg:=DOM Creer ref XML("svg";"http://www.w3.org/2000/svg")
$ref:=DOM Creer element XML($svg;"text";"font-size";26;"fill";"red")
DOM ECRIRE ATTRIBUT XML($ref;"y";"1em")
DOM ECRIRE VALEUR ELEMENT XML($ref;"Hello World")
SVG EXPORTER VERS IMAGE($svg;vImage;Copier source données XML)
DOM FERMER XML($svg)
```

SVG FIXER ATTRIBUT

SVG FIXER ATTRIBUT ({* ;} objetImage ; id_Element ; nomAttribut ; valeurAttribut {; nomAttribut2 ; valeurAttribut2 ; ... ; nomAttributN ; valeurAttributN} {; *})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objetImage est un nom d'objet (chaîne) Si omis, objetImage est une variable ou un champ
objetImage	Image	⇒	Nom d'objet (si * spécifié) ou Variable ou champ (si * omis)
id_Element	Texte	⇒	ID de l'élément dont un ou plusieurs attribut(s) sont à définir
nomAttribut	Chaîne	⇒	Attribut à définir
valeurAttribut	Chaîne, Entier long	⇒	Nouvelle valeur d'attribut
*	Opérateur	⇒	Si passé = modifier l'arbre DOM interne de l'image SVG (variable uniquement)

Description

La commande **SVG FIXER ATTRIBUT** permet de modifier la valeur d'un attribut existant dans l'arbre de rendu SVG d'une image affichée ou dans l'arbre DOM interne d'une image.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *objetImage* est un nom d'objet (une chaîne). Dans ce cas, la commande s'applique aux paramètres de l'image de rendu attachée à l'objet (à noter que les paramètres et donc l'image de rendu de l'objet ne sont créés que si la commande **SVG FIXER ATTRIBUT** est appelée au moins une fois).

Si vous ne passez pas le premier paramètre *, vous indiquez que le paramètre *objetImage* est une variable ou un champ. Vous ne passez alors pas une chaîne mais une référence de variable (variable objet uniquement) ou de champ. Dans ce cas, la commande s'applique aux images de rendu de tous les objets qui utilisent la variable ou le champ.

Par défaut, les modifications effectuées par cette commande s'appliquent uniquement aux images de rendu, elle ne sont pas stockées dans la source de données (l'arbre DOM interne) et sont perdues lorsque l'image est effacée par programmation ou lorsque le formulaire est fermé. Il est toutefois possible de reporter ces modifications dans l'arbre DOM interne de l'image lorsque le paramètre *objetImage* référence une variable : il suffit pour cela de passer un second * en dernier paramètre. Ce principe permet de préserver des modifications effectuées à la volée.

Notes :

- Le report des modifications dans l'arbre DOM interne n'est pas possible lorsque le paramètre *objetImage* référence un objet.
- Pour que le report des modifications soit possible, la variable SVG doit avoir été créée à partir d'un document DOM (avec **DOM EXPORTER VERS VARIABLE**). Si la variable SVG a été créée à partir d'un fichier et si vous passez le second paramètre *, la commande ne fera rien et une erreur sera générée car dans ce cas la source de données ne contient pas de document DOM modifiable.
- Pour modifier la source de données d'une image SVG, vous pouvez également utiliser les commandes **XML DOM** ou le composant **MissingRef** fourni par 4D.

Le paramètre *id_Element* permet de définir l'ID (attribut "id" ou "xml:id") de l'élément dont vous souhaitez modifier un ou plusieurs attribut(s).

Passez dans les paramètres *nomAttribut* et *valeurAttribut* respectivement l'attribut à écrire et sa valeur (sous forme de variables, champs ou valeurs littérales). Vous pouvez passer autant de couples attribut/valeur que vous voulez.

La commande **SVG FIXER ATTRIBUT** vous permet de modifier (mais pas d'ajouter ou de supprimer) la plupart des attributs SVG, comme par exemple 'fill', 'opacity', 'font-family', etc. Pour une définition complète des attributs SVG, reportez-vous aux documents de référence disponibles sur Internet, par exemple http://www.w3.org/TR/SVG11/attindex.html. La mise à jour de l'image de rendu est immédiate, les modifications sont reportées en cascade sur les éléments enfants pour les styles héritables.

A noter que pour des raisons techniques, les attributs de certains éléments ainsi que certains attributs ne sont pas modifiables. Le tableau suivant liste les éléments modifiables, les éléments non modifiables ainsi que les attributs non modifiables :

Eléments dont les attributs sont modifiables

svg	Restrictions : - "width" et "height" ne sont pas modifiables (1) - "viewBox" n'est modifiable que si "width" et "height" sont définis dans le document d'origine
g	
defs	
use	
filter	Restriction : les éléments enfants fe_xxx ne sont pas modifiables
circle	
ellipse	
line	
polyline	
polygon	
path	
rect	
text, tspan, textArea	L'attribut spécifique "4d-text" vous permet de modifier le texte d'un élément "text", "tspan" ou "textArea" (cf. exemple)
Image	

Eléments dont les attributs ne sont pas modifiables

linearGradient, radialGradient, Stop, solidColor, marker, symbol, clipPath, filter et les éléments commençant par fe, style, pattern	Cet ensemble désigne tous les éléments référençables ou contenus dans un élément référençable. Cela signifie qu'il n'est pas possible par exemple de redéfinir les attributs d'un gradient (mais il est possible de changer le gradient utilisé). De même, pour changer un marqueur de couleur noire en marqueur rouge, il faudra définir deux marqueurs dans le document SVG (un noir et un rouge) et sélectionner l'un ou l'autre. Il n'est pas possible non plus par exemple de modifier la couleur d'un rectangle s'il a pour parent un élément symbol ou marker

Attributs non modifiables

id ou xml:id	
lang ou xml:lang	
class ou xml:class	
width, height	Concerne les attributs de l'élément 'svg' uniquement (1)

(1) Ces attributs ne peuvent être modifiés car ils définissent et structurent l'image résultante. Les attributs *width* et *height* de l'élément *svg* servent à définir les dimensions initiales de l'image dans 4D et ces dimensions doivent rester constantes après la création de l'image (il est toutefois possible de modifier les dimensions de l'image résultante avec la commande **TRANSFORMER IMAGE** de 4D).

Reportez-vous également à la description de la commande **SVG LIRE ATTRIBUT** pour obtenir la liste des attributs 4D réservés et dédiés à l'animation.

Si vous tentez de modifier un attribut d'un élément non pris en charge ou l'un de ses enfants, la commande ne fait rien et aucune erreur n'est générée.

Si la commande est exécutée en-dehors du contexte d'un formulaire ou si un *objetImage* invalide est passé, la variable OK prend la valeur 0. Si la commande a été exécutée correctement, elle prend la valeur 1.

Exemple

Modification du contenu d'un élément de type texte :

```
SVG FIXER ATTRIBUT(*;nom_objet_image;text_element_ID;"4d-text";"Ceci est un texte")
```

Note : Il n'y a pas de *namespace* pour que l'attribut puisse être utilisé dans une feuille de style CSS sans risque de conflit.

SVG LIRE ATTRIBUT

SVG LIRE ATTRIBUT ({* ;} objetImage ; id_Element ; nomAttribut ; valeurAttribut)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objetImage est un nom d'objet (chaîne) Si omis, objetImage est une variable ou un champ
objetImage	Image	⇒	Nom d'objet (si * spécifié) ou Variable ou champ (si * omis)
id_Element	Texte	⇒	ID de l'élément dont vous souhaitez connaître une valeur d'attribut
nomAttribut	Chaîne	⇒	Nom d'attribut
valeurAttribut	Chaîne, Entier long	⇐	Valeur courante de l'attribut

Description

La commande **SVG LIRE ATTRIBUT** permet de lire la valeur courante de l'attribut *nomAttribut* dans un objet ou une image SVG.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objetImage* est un nom d'objet (une chaîne). Dans ce cas, la commande retourne la valeur de l'attribut pour l'image de rendu attachée à l'objet. Cette valeur peut avoir été modifiée par **SVG FIXER ATTRIBUT** par exemple.
Si vous ne passez pas le paramètre *, vous indiquez que le paramètre *objetImage* est une variable ou un champ. Vous ne passez alors pas une chaîne mais une référence de variable (variable objet uniquement) ou de champ. Dans ce cas, la commande retourne la valeur de l'attribut pour l'image de rendu initiale (correspondant à la source de données de la variable).

Note : Ce principe s'applique également à la commande **SVG Chercher ID element par coordonnees**.

Le paramètre *id_Element* permet de définir l'ID (attribut "id" ou "xml:id") de l'élément dont vous souhaitez lire la valeur d'attribut.

Pour plus d'informations sur les attributs SVG, reportez-vous à la description de la commande **SVG FIXER ATTRIBUT**. Voici la liste des attributs 4D réservés et dédiés à l'animation :

Attributs	Accès	Commentaire
4D-text	lecture/écriture	Remplace/lit le contenu du noeud de texte. Utilisable avec les éléments 'text', 'tspan' et 'textArea'
4D-bringToFront	écriture	Si 'true', déplacer le noeud devant les noeuds frères. Utilisable uniquement avec la commande **SVG FIXER ATTRIBUT**
4D-isOfClass-{IDENT [[S|COMMA] IDENT]*}	lecture	Si l'attribut de la classe héritée du noeud contient tous les noms de classes, retourne 'true' sinon retourne 'false'. Retourne par exemple true pour "4D-isOfClass-land" si la classe héritée du noeud est "land department01")
4D-enableD2D	lecture/écriture	Si 'false', inactive Direct2D pour le moteur de rendu SVG. En effet, les filtres SVG ne sont pas rendus en Direct2D mais ils le sont en GDI/GDIPlus. Cette option permet de bénéficier des filtres SVG même si la base est en Direct2D. A noter que cette option n'est prise en compte que si *objetImage* contient déjà une image chargée. En revanche, elle n'a besoin d'être définie qu'une seule fois par session (par exemple avec un SVG simple chargé en mémoire depuis une variable texte au démarrage de la base) car elle est globale au moteur.

SVG MONTRER ELEMENT

SVG MONTRER ELEMENT ({* ;} objetImage ; id {; marge})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objetImage est un nom d'objet (chaîne) Si omis, objetImage est une variable ou un champ
objetImage	Image	⇒	Nom d'objet (si * spécifié) ou Variable ou champ (si * omis)
id	Texte	⇒	Attribut id de l'élément à visualiser
marge	Entier long	⇒	Marge de visibilité (en pixels par défaut)

Description

La commande **SVG MONTRER ELEMENT** déplace le document SVG *objetImage* de façon à rendre visible l'élement dont l'attribut "id" est désigné par le paramètre *id*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objetImage* est un nom d'objet (une chaîne). Dans ce cas, la commande s'applique à l'image de rendu attachée à l'objet. Si vous ne passez pas le paramètre *, vous indiquez que le paramètre *objetImage* est une variable ou un champ. Vous ne passez alors pas une chaîne mais une référence de variable (variable objet uniquement) ou de champ. Dans ce cas, la commande s'applique aux images de rendu de tous les objets qui utilisent la variable ou le champ (mais pas à l'image de rendu initiale).

La commande déplace le document SVG de manière à ce que la totalité de l'objet, dont les limites sont définies par son rectangle englobant, soit visible. Le paramètre *marge* permet de configurer l'amplitude du déplacement en définissant la distance devant séparer l'objet visualisé des bords du document. Autrement dit, le rectangle englobant sera augmenté de *marge* pixels en largeur et en hauteur. Par défaut, la valeur de déplacement est de 4 pixels.

Cette commande n'a d'effet qu'en mode d'affichage "top left" (avec barres de défilement).

Si la commande est exécutée en-dehors du contexte d'un formulaire ou si un *objetImage* invalide est passé, la variable OK prend la valeur 0. Si la commande a été exécutée correctement, elle prend la valeur 1.

Table

- PAS DE TABLE PAR DEFAUT
- Table du formulaire courant
- TABLE PAR DEFAUT
- Table par defaut courante

PAS DE TABLE PAR DEFAUT

PAS DE TABLE PAR DEFAUT
Ne requiert pas de paramètre

Description

La commande **PAS DE TABLE PAR DEFAUT** permet d'annuler l'effet de la commande **TABLE PAR DEFAUT**. Après l'exécution de cette commande, il n'y a plus de table par défaut définie pour le process.
Si la commande **TABLE PAR DEFAUT** n'avait pas été appelée au préalable, cette commande ne fait rien.

Cette commande est liée à l'utilisation de formulaires projets (formulaires non liés à des tables) : la plupart des commandes relatives aux formulaires (hors formulaires utilisateurs) acceptent un paramètre facultatif de type table comme premier paramètre. C'est par exemple le cas des commandes **FORM LIRE PARAMETRE**, **Creer fenetre formulaire** ou **DIALOGUE**. Comme un formulaire projet et un formulaire table peuvent avoir le même nom, ce paramètre permet de déterminer le formulaire à utiliser : passez le paramètre lorsque vous souhaitez adresser un formulaire table et ne le passez pas dans le cas d'un formulaire projet.
Dans une base contenant un formulaire projet nommé "LeForm" et un formulaire table du même nom pour la table [Table1] :

```
DIALOGUE([Table1];"LeForm") `4D utilise le formulaire table
DIALOGUE("LeForm") `4D utilise le formulaire projet
```

Ce principe est toutefois caduc lorsque la commande **TABLE PAR DEFAUT** a été exécutée et que la base contient un formulaire projet et un formulaire table du même nom. En effet, dans ce cas 4D utilisera le formulaire table de la table par défaut, même si le paramètre *laTable* n'est pas passé. Dans ce cas, pour permettre l'utilisation de formulaires projet, il suffit d'exécuter la commande **PAS DE TABLE PAR DEFAUT**.

Exemple

Dans une base contenant un formulaire projet nommé "LeForm" et un formulaire table du même nom pour la table [Table1] :

```
TABLE PAR DEFAUT([Table1])
DIALOGUE("LeForm") `4D utilise le formulaire table
PAS DE TABLE PAR DEFAUT
DIALOGUE("LeForm") `4D utilise le formulaire projet
```

Table du formulaire courant

Table du formulaire courant -> Résultat

Paramètre	Type	Description
Résultat	Pointeur	Pointeur vers la table à laquelle appartient le formulaire actuellement affiché

Description

La fonction **Table du formulaire courant** retourne un pointeur vers la table à laquelle appartient le formulaire affiché à l'écran ou imprimé dans le process courant.

La fonction retourne Nil dans les cas suivants :

- il n'y a pas de formulaire affiché ou en cours d'impression dans le process courant,
- le formulaire courant est un formulaire projet.

Si plusieurs fenêtres sont ouvertes simultanément dans le process courant (ce qui signifie que la dernière fenêtre ouverte est la fenêtre courante active), la fonction retourne un pointeur vers la table du formulaire affiché dans la fenêtre active.

Si le formulaire affiché est le formulaire détaillé d'une zone de sous-formulaire (c'est-à-dire que pendant la saisie de données, l'utilisateur a double-cliqué sur un enregistrement ou un sous-enregistrement dans une zone de sous-formulaire "double-cliquable"), la fonction retourne :

- un pointeur vers la table de la zone de sous-formulaire, si cette dernière affiche une table.
- un pointeur non significatif si la zone de sous-formulaire affiche une sous-table.

Exemple

Dans votre application, vous utilisez la convention suivante : au moment de l'affichage d'un enregistrement, la variable *vsEnrCourant*, si elle est présente dans un formulaire, affiche "Nouvel enregistrement" si vous créez un nouvel enregistrement, ou par exemple "56 parmi 5200" si vous ouvrez le 56e enregistrement d'une sélection en comportant 5200. Pour cela, vous pouvez créer une fois la variable *vsEnrCourant* et lui associer la méthode objet décrite ci-dessous, puis la copier et la coller dans tous les formulaires que vous voulez :

```
  //Méthode objet de la variable non saisissable vsEnrCourant
 Au cas ou
    :(Evenement formulaire=Sur chargement)
       C_ALPHA(31;vsEnrCourant)
       C_POINTEUR($vpTableParente)
       C_ENTIER LONG($vlNumEnr)
       $vpTableParente:=Table du formulaire courant
       $vlNumEnr:=Numero enregistrement($vpTableParente->)
       Au cas ou
          :($vlNumEnr=-3)
             vsEnrCourant:="Nouvel enregistrement"
          :($vlNumEnr=-1)
             vsEnrCourant:="Pas d'enregistrement"
          :($vlNumEnr>=0)
             vsEnrCourant:=Chaine(Numero dans selection($vpTableParente->))+" parmi "+Chaine(Enregistrements
trouves($vpTableParente->))
       Fin de cas
 Fin de cas
```

TABLE PAR DEFAUT

TABLE PAR DEFAUT (laTable)

Paramètre	Type		Description
laTable	Table	➡	Table à définir comme table par défaut

Description

Conseil : Bien que l'appel de **TABLE PAR DEFAUT** et l'omission du nom de la table rendent le code plus lisible, la plupart des programmeurs estiment que l'utilisation de cette commande apporte plus d'inconvénients que d'avantages.
En particulier, notez que **TABLE PAR DEFAUT** est prioritaire lorsque vous utilisez par exemple la commande **DIALOGUE** avec un formulaire projet et qu'un formulaire de la table par défaut a le même nom.

TABLE PAR DEFAUT désigne *laTable* comme la table par défaut pour le process courant.

Un process n'a pas de table par défaut tant que la commande **TABLE PAR DEFAUT** n'a pas été exécutée. Après qu'une table par défaut ait été désignée, toute commande pour laquelle le paramètre *laTable* n'a pas été défini s'appliquera à la table par défaut. Considérez par exemple l'instruction suivante :

```
FORM FIXER ENTREE([maTable];"Formulaire")
```

Si [maTable] a préalablement été définie comme table par défaut, la même instruction pourrait s'écrire :

```
FORM FIXER ENTREE("Formulaire")
```

Une des raisons pour lesquelles vous pouvez définir une table par défaut est l'écriture de code qui ne soit pas lié à une table. Cela permet au même code d'être appliqué à différentes tables.
Vous pouvez aussi utiliser des pointeurs vers des tables pour écrire du code non lié aux tables. Pour plus d'informations sur cette technique, reportez-vous à la description de la commande **Nom de la table**.

TABLE PAR DEFAUT ne permet pas d'omettre les noms de tables lorsque vous vous référez à des champs. Par exemple :

```
[MaTable]MonChamp:="Une chaîne" ` OK
```

ne peut pas s'écrire :

```
TABLE PAR DEFAUT([MaTable])
MonChamp:="Une chaîne" ` Incorrect
```

... simplement parce qu'une table par défaut a été définie. Toutefois, vous pouvez omettre le nom de la table lorsque vous vous référez à des champs dans des triggers, des formulaires et des objets appartenant à la table.

Dans 4D, toutes les tables sont “ouvertes” et prêtes à être utilisées. **TABLE PAR DEFAUT** n'ouvre pas de table, ne définit pas de table courante et ne prépare pas de table pour la saisie ou l'affichage. **TABLE PAR DEFAUT** est simplement une facilité de programmation proposée pour accélérer la saisie du code et le rendre plus facile à lire.

Exemple

L'exemple suivant présente la même méthode avec et sans la commande **TABLE PAR DEFAUT**. Le code est une boucle souvent utilisée pour créer de nouveaux enregistrements dans une base. Les commandes **FORM FIXER ENTREE** et **AJOUTER ENREGISTREMENT** nécessitent le nom d'une table comme premier paramètre :

```
FORM FIXER ENTREE([Clients];"Ajout Enrg")
Repeter
   AJOUTER ENREGISTREMENT([Clients])
Jusque(OK=0)
```

Voici le résultat lorsqu'une table par défaut est définie :

```
TABLE PAR DEFAUT([Clients])
FORM FIXER ENTREE("Ajout Enrg")
Repeter
   AJOUTER ENREGISTREMENT
Jusque(OK=0)
```

Table par defaut courante

Table par defaut courante -> Résultat

Paramètre	Type	Description
Résultat	Pointeur	Pointeur vers la table par défaut

Description

Table par defaut courante retourne un pointeur vers la table qui a été passée au dernier appel de la commande **TABLE PAR DEFAUT** pour le process courant.

Exemple

La ligne de code suivante inscrit le nom de la table courante par défaut dans le titre de la fenêtre :

```
CHANGER TITRE FENETRE(Nom de la table(Table par defaut courante))
```

Tableaux

- Présentation des tableaux
- Créer des tableaux
- Tableaux et objets de formulaire
- Les tableaux et le langage 4D
- Tableaux et pointeurs
- Utiliser l'élément zéro d'un tableau
- Tableaux à deux dimensions
- Tableaux et mémoire
- AJOUTER A TABLEAU
- Chercher dans tableau
- Chercher dans tableau trié
- Compter dans tableau
- COPIER TABLEAU
- INSERER DANS TABLEAU
- LISTE VERS TABLEAU
- SELECTION LIMITEE VERS TABLEAU
- SELECTION VERS TABLEAU
- SUPPRIMER DANS TABLEAU
- TABLEAU BLOB
- TABLEAU BOOLEEN
- TABLEAU BOOLEEN SUR ENSEMBLE
- TABLEAU DATE
- TABLEAU ENTIER
- TABLEAU ENTIER LONG
- TABLEAU ENTIER LONG SUR SELECTION
- TABLEAU HEURE
- TABLEAU IMAGE
- TABLEAU MULTI TRI
- TABLEAU OBJET
- TABLEAU POINTEUR
- TABLEAU REEL
- TABLEAU TEXTE
- TABLEAU VERS LISTE
- TABLEAU VERS SELECTION
- Taille tableau
- TEXTE VERS TABLEAU
- TRIER TABLEAU
- VALEURS DISTINCTES
- VALEURS DISTINCTES ATTRIBUT Nouveauté 16.0
- VALEURS DISTINCTES CHEMINS Nouveauté 16.0
- *_o_TABLEAU ALPHA*

Présentation des tableaux

Un **tableau** est une série ordonnée de variables de même type. Chaque variable est appelée un **élément** du tableau. La **taille** d'un tableau est le nombre d'éléments qu'il contient. La taille du tableau doit être définie au moment de sa création ; vous pouvez ensuite la modifier aussi souvent que nécessaire en ajoutant, insérant, ou supprimant des éléments, ou en appelant de nouveau la commande que vous avez utilisée pour créer le tableau.

Vous créez un tableau au moyen de l'une des commandes de déclaration de tableau. Pour plus d'informations, reportez-vous à la section **Créer des tableaux**.

Les éléments sont numérotés de **1 à N**, où N est la taille du tableau. Un tableau a toujours un **élément zéro**, auquel vous pouvez accéder tout comme vous accédez à n'importe quel autre élément du tableau, mais cet élément n'est pas affiché lorsqu'un tableau est utilisé dans un formulaire. Rien ne vous empêche cependant de l'utiliser avec le langage. Pour plus d'informations sur l'élément zéro, reportez-vous à la section **Utiliser l'élément zéro d'un tableau**.

Les tableaux sont des variables 4D. Comme toute variable, un tableau a une portée et suit les règles du langage 4D, bien qu'il existe quelques différences spécifiques. Pour plus d'informations, reportez-vous aux sections **Les tableaux et le langage 4D** et **Tableaux et pointeurs**.

Les tableaux sont des objets du langage : vous pouvez créer et utiliser des tableaux qui n'apparaîtront jamais à l'écran. Mais les tableaux sont également des objets d'interface utilisateur. Pour plus d'informations sur l'interaction entre les tableaux et les objets de formulaire, reportez-vous à la section **Tableaux et objets de formulaire**.

Les tableaux doivent être utilisés pour manipuler une certaine quantité de données pendant une période brève. En contrepartie, comme les tableaux résident en mémoire, ils sont d'une utilisation rapide et facile. Pour plus d'informations, reportez-vous à la section **Tableaux et mémoire**.

Créer des tableaux

Vous créez un tableau au moyen de l'une des commandes de déclaration de tableau décrites dans ce chapitre. Voici la liste des commandes de déclaration de tableau :

Commande	Crée ou redimensionne un tableau de :
TABLEAU ENTIER	Entiers (sur 2 octets)
TABLEAU ENTIER LONG	Entiers (sur 4 octets)
TABLEAU REEL	Réels
TABLEAU TEXTE	Textes (jusqu'à 2 Go de texte par élément)*
_o_TABLEAU ALPHA	Textes (obsolète)*
TABLEAU DATE	Dates
TABLEAU BOOLEEN	Booléens
TABLEAU IMAGE	Images
TABLEAU POINTEUR	Pointeurs
TABLEAU OBJET	Objets de langage
TABLEAU BLOB	BLOBs
TABLEAU HEURE	Heures

Chaque commande de déclaration de tableau peut créer ou redimensionner des tableaux à une ou à deux dimensions. Pour plus d'informations sur les tableaux à deux dimensions, reportez-vous à la section **Tableaux à deux dimensions**.

(*) Il n'y a aucune différence entre les tableaux Texte et les tableaux Alpha. Le paramètre *longueurChaîne* de la commande **_o_TABLEAU ALPHA** est ignoré. Il est conseillé d'utiliser des tableaux Texte. La commande **_o_TABLEAU ALPHA** est conservée pour des raisons de compatibilité uniquement.

Cette ligne de code crée (déclare) un tableau d'entiers de 10 éléments :

```
TABLEAU ENTIER(aiUnTableau;10)
```

Ensuite, cette ligne de code redimensionne le même tableau à 20 éléments :

```
TABLEAU ENTIER(aiUnTableau;20)
```

Enfin, cette ligne de code redimensionne le même tableau à 0 élément :

```
TABLEAU ENTIER(aiUnTableau;0)
```

Vous référencez les éléments d'un tableau en utilisant des accolades ({…}). Un nombre entre accolades donne accès à l'adresse d'un élément particulier. Ce nombre est appelé **numéro de l'élément**. L'exemple ci-dessous place cinq noms dans le tableau nommé *atNoms* et les affiche ensuite dans une fenêtre d'alerte :

```
TABLEAU TEXTE(atNoms;5)
atNoms{1}:="Richard"
atNoms{2}:="Sarah"
atNoms{3}:="Pierre"
atNoms{4}:="Martine"
atNoms{5}:="Jean"
Boucle($vlElem;1;5)
   ALERTE("L'élément #"+Chaine($vlElem)+" est égal à: "+atNoms{$vlElem})
Fin de boucle
```

Notez la syntaxe *atNoms{$vlElem}*. Au lieu de spécifier un nombre littéral comme *atNoms{3}*, vous pouvez utiliser une variable numérique indiquant à quel élément d'un tableau vous accédez.

Si vous utilisez les itérations permises par les structures répétitives (**Boucle...Fin de boucle**, **Repeter...Jusque** ou **Tant que...Fin tant que**), vous pouvez accéder à tout ou partie des éléments d'un tableau avec très peu de code.

Les tableaux et les autres commandes du langage 4D

Il existe d'autres commandes 4D qui permettent de créer ou de manipuler des tableaux. En particulier :

- Pour travailler avec des tableaux et des sélections d'enregistrements, utilisez les commandes **SELECTION LIMITEE VERS TABLEAU**, **SELECTION VERS TABLEAU**, **TABLEAU VERS SELECTION** et **VALEURS DISTINCTES**.
- Les objets de type List box sont basés sur les tableaux ; plusieurs des commandes du thème “List box” manipulent des tableaux, par exemple **LISTBOX INSERER LIGNES**.
- Vous pouvez créer des graphes à partir de valeurs stockées dans des tables et des tableaux. Pour plus d'informations, reportez-vous à la commande **GRAPHE**.
- Les commandes **LISTE VERS TABLEAU** et **TABLEAU VERS LISTE**.
- De nombreuses commandes peuvent construire des tableaux en un appel, par exemple **LISTE DES POLICES**, **LISTE FENETRES**, **LISTE DES VOLUMES**, **LISTE DES DOSSIERS**, **LISTE DES DOCUMENTS**, **LIRE CORRESPONDANCE PORT SERIE**, **SAX LIRE ELEMENT XML**, etc.

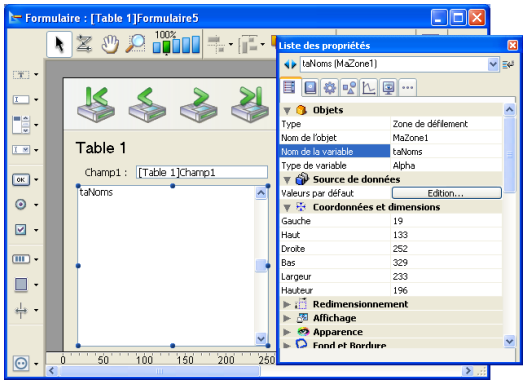
Les tableaux sont des objets de langage — vous pouvez créer et utiliser des tableaux qui n'apparaîtront jamais à l'écran. Cependant, les tableaux sont aussi des objets d'interface utilisateur. Voici les types d'**objets de formulaire** gérés par des tableaux :

- Pop-up/Liste déroulante
- Combo Box
- Zone de défilement (obsolète à compter de 4D v13)
- Onglet
- List box

Si vous pouvez prédéfinir ces objets dans l'éditeur de formulaires en utilisant le bouton des valeurs par défaut de la Liste des propriétés (hormis les List box), vous pouvez également les définir par programmation, en utilisant les commandes de tableaux. Dans les deux cas, l'objet formulaire est géré par un tableau, créé par vous ou par 4D.

En utilisant ces objets, vous pouvez détecter quel élément de l'objet a été sélectionné (ou a reçu un clic souris) en testant l'**élément sélectionné** du tableau. Inversement, vous pouvez sélectionner un élément de l'objet en désignant l'élément de tableau correspondant.

Quand un tableau est utilisé pour gérer un objet de formulaire, il a une nature double ; il est à la fois un objet de langage et un objet d'interface utilisateur. Par exemple, créez un formulaire, dans lequel vous placez une zone de défilement :



Le nom de la variable associée, ici *taNoms*, est le nom du tableau que vous utilisez pour créer et gérer la zone de défilement.

Notes :

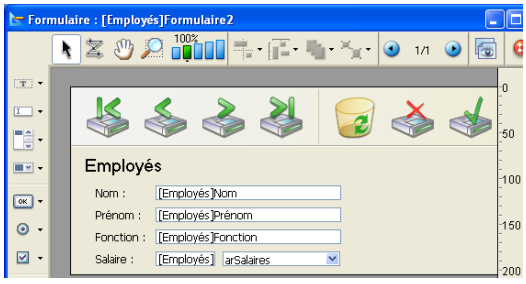
- L'**élément sélectionné** du tableau est stocké en interne dans une variable de type entier. Par conséquent, il n'est pas possible de l'utiliser avec des tableaux comportant plus de 32767 éléments (un tel nombre d'éléments n'est toutefois pas approprié pour l'affichage sous forme d'objet de formulaire).
- Vous ne pouvez pas afficher des tableaux à deux dimensions ni des tableaux de pointeurs.
- La gestion des objets de type **List box** (pouvant contenir plusieurs tableaux) revêt de nombreux aspects particuliers. Ces spécificités sont traitées dans la section **Gestion programmée des objets de type List box**.

Exemple : création d'une liste déroulante

L'exemple suivant montre comment remplir un tableau et l'afficher dans une liste déroulante. Un tableau *arSalaires* est créé au moyen de la commande **TABLEAU REEL**. Il contient tous les salaires des personnes dans une entreprise américaine. Lorsque l'utilisateur sélectionne un élément dans la liste déroulante, le champ *[Personnel]Salaire* reçoit la valeur choisie.

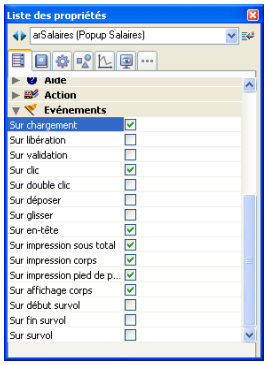
Création de la liste déroulante arSalaires dans un formulaire

Créez une liste déroulante et nommez-la *arSalaires*. Le nom de la liste déroulante doit être le même que celui du tableau.

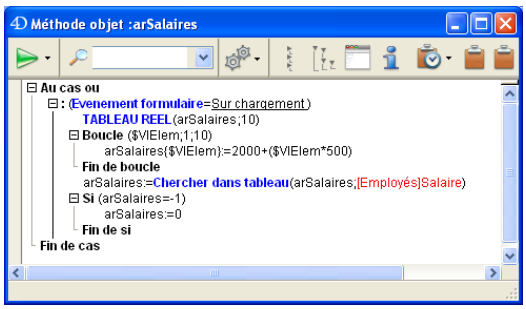


Initialisation du tableau

Initialisez le tableau *arSalaires* en spécifiant l'événement Sur chargement pour l'objet. Pour cela, n'oubliez pas d'activer cet événement dans la Liste des propriétés, comme illustré ci-dessous :



Cliquez sur le bouton **Méthode objet...** et écrivez la méthode suivante :



Les lignes :

```
TABLEAU REEL(arSalaires;10)
Boucle($vlElem;1;10)
   arSalaires{$vlElem}:=2000+($vlElem*500)
Fin de boucle
```

... créent le tableau numérique 2500, 3000... 7000, correspondant aux salaires annuels allant de $30 000 à $84 000, avant impôts.

Les lignes :

```
arSalaires:=Chercher dans tableau(arSalaires;[Employés]Salaire)
Si(arSalaires=-1)
   arSalaires:=0
Fin de si
```

... gèrent à la fois la création d'un nouvel enregistrement et la modification d'un enregistrement existant.

- Si vous créez un nouvel enregistrement, le champ *[Employés]Salaire* est initialement égal à zéro. Dans ce cas, **Chercher dans tableau** ne trouve pas la valeur dans le tableau et retourne -1. Le test **Si (arSalaires=-1)** remet *arSalaires* à zéro, indiquant qu'aucun élément n'est sélectionné dans la liste déroulante.
- Si vous modifiez un enregistrement existant, **Chercher dans tableau** récupère la valeur dans le tableau et affecte à l'élément sélectionné de la liste déroulante la valeur courante du champ. Si la valeur pour un employé n'est pas dans la liste, le test **Si (arSalaires=-1)** désélectionne tous les éléments de la liste.

Note : Pour plus d'informations concernant l'**élément de tableau sélectionné**, lisez les paragraphes suivants.

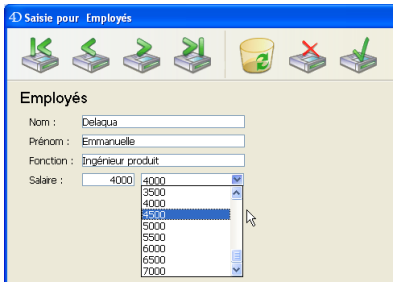
Assignation de la valeur sélectionnée au champ [Employés]Salaire

Pour reporter la valeur sélectionnée dans la liste déroulante *arSalaires*, il vous suffit de gérer l'événement Sur clic de l'objet. Le numéro de l'élément sélectionné est la valeur du tableau *arSalaires*. En conséquence, l'expression *arSalaires{arSalaires}* retourne la valeur choisie dans la liste déroulante.

Complétez ainsi la méthode de l'objet *arSalaires* :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TABLEAU REEL(arSalaires;10)
      Boucle($vlElem;1;10)
         arSalaires{$vlElem}:=2000+($vlElem*500)
      Fin de boucle
      arSalaires:=Chercher dans tableau(arSalaires;[Employés]Salaire)
      Si(arSalaires=-1)
         arSalaires:=0
      Fin de si
   :(Evenement formulaire=Sur clic)
      [Employés]Salaire:=arSalaires{arSalaires}
Fin de cas
```

En exécution, la liste déroulante se présente comme suit :



Les paragraphes suivants décrivent les opérations élémentaires que vous pouvez effectuer sur les tableaux lorsque vous les utilisez comme objets de formulaire.

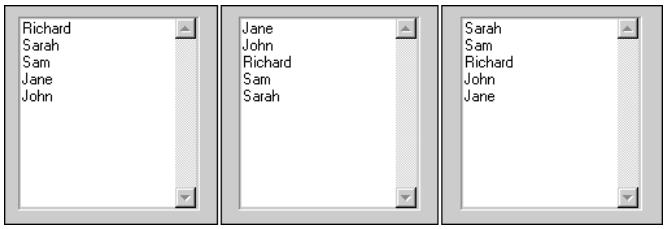
Obtenir la taille d'un tableau

Vous pouvez obtenir la taille courante d'un tableau au moyen de la commande **Taille tableau**. Si on reprend l'exemple précédent, la ligne de code qui suit affichera 5:

```
ALERTE("La taille du tableau taNoms est: "+Chaine(Taille tableau(taNoms)))
```

Trier (réordonner) les éléments du tableau

Vous pouvez réordonner les éléments d'un tableau au moyen de la commande **TRIER TABLEAU** ou de plusieurs tableaux à l'aide de la commande **TABLEAU MULTI TRI**. Si on reprend l'exemple précédent, et étant entendu que le tableau est affiché comme une liste déroulante, vous pourrez voir ceci :



(a) Sur la gauche, la liste telle qu'elle se présente initialement.

(b) Au centre, la liste telle qu'elle se présente après l'exécution de la ligne de code suivante :

```
TRIER TABLEAU(taNoms;>)
```

(c) Sur la droite, la liste telle qu'elle se présente après l'exécution de la ligne de code suivante :

```
TRIER TABLEAU(taNoms;<)
```

Ajouter et supprimer des éléments

Vous pouvez ajouter, insérer, ou supprimer des éléments de tableau au moyen des commandes **AJOUTER A TABLEAU**, **INSERER DANS TABLEAU** et **SUPPRIMER DANS TABLEAU**.

Gestion des clics dans un tableau : test de l'élément sélectionné

Si on reprend l'exemple précédent, étant entendu que le tableau est affiché en tant que liste déroulante, vous pouvez gérer les clics souris de la manière suivante :

```
   ` Méthode objet liste déroulante taNoms
Au cas ou
   :(Evenement formulaire=Sur chargement)
   ` Initialiser le tableau (comme vu précédemment)
      TABLEAU TEXTE(taNoms;5)
   ` ...
   :(Evenement formulaire=Sur libération)
   ` Nous n'avons plus besoin du tableau
      EFFACER VARIABLE(taNoms)

   :(Evenement formulaire=Sur clic)
      Si(taNoms#0)
         vtInfo:="Vous avez cliqué sur : "+taNoms{taNoms}
      Fin de si
   :(Evenement formulaire=Sur double clic)
      Si(taNoms#0)
         ALERTE("Vous avez double-cliqué sur : "+taNoms{taNoms})
      Fin de si
Fin de cas
```

Note : Les événements doivent avoir été activés dans les propriétés de l'objet.

Alors que la syntaxe *taNoms{$vlElem}* vous permet de travailler sur un élément particulier du tableau, la syntaxe *taNoms* retourne le numéro de l'élément

sélectionné dans le tableau. Ainsi, la syntaxe *taNoms{taNoms}* signifie “la valeur de l'élément sélectionné dans le tableau *taNoms*.” Si aucun élément n'est sélectionné, *taNoms* est égal à 0 (zéro). Le test **Si (taNoms#0)** détecte si un élément est effectivement sélectionné ou non.

Désigner l'élément sélectionné

Vous pouvez changer par programmation l'élément sélectionné en assignant une valeur au tableau.

Exemples

```
 ` Sélectionner le premier élément (si le tableau n'est pas vide)
taNoms:=1

 ` Sélectionner le dernier élément (si le tableau n'est pas vide)
taNoms:=Taille tableau(taNoms)

 ` Déselectionner l'élément sélectionné (s'il y en a un), aucun élément n'est alors sélectionné
taNoms:=0

Si((0<taNoms)&(taNoms<Taille tableau(taNoms))
 ` Si possible, sélectionner l'élément suivant l'élément sélectionné
   taNoms:=taNoms+1
Fin de si

Si(1<taNoms)
 ` Si possible, sélectionner l'élément précédent l'élément sélectionné
   taNoms:=taNoms-1
Fin de si
```

Recherche d'une valeur dans le tableau

La commande **Chercher dans tableau** recherche une valeur particulière dans un tableau. Si nous reprenons l'exemple précédent, voici le code qui sélectionnera l'élément dont la valeur est “Richard”, si c'est ce que vous saisissez dans la boîte de dialogue de demande :

```
$vsNom:=Demander("Saisissez un prénom :")
Si(OK=1)
   $vlElem:=Chercher dans tableau(taNoms;$vsNom)
   Si($vlElem>0)
      taNoms:=$vlElem
   Sinon
      ALERTE("Il n'y a pas de "+$vsName+" dans cette liste de prénoms.")
   Fin de si
Fin de si
```

Les pop-up menus, listes déroulantes, zones de défilement et les onglets peuvent généralement être gérés de la même manière. Bien entendu, aucun code supplémentaire n'est nécessaire pour le redessinement des objets à l'écran à chaque fois que vous modifiez la valeur d'un élément, en ajoutez ou en supprimez.

Note : Pour créer et utiliser des onglets avec des icônes ainsi que des onglets activés ou désactivés, vous devez utiliser une liste hiérarchique comme objet de gestion associé à l'onglet. Pour plus d'informations, reportez-vous à l'exemple de la commande **Nombre de process**.

Gestion des combo boxes

Alors que vous pouvez gérer au moyen des algorithmes décrits dans la section précédente les pop-up menus, les listes déroulantes, les zones de défilement et les onglets, vous devez gérer les combo boxes différemment.

Une combo box est en réalité une zone de texte saisissable à laquelle est rattachée une liste de valeurs prédéfinies (les éléments d'un tableau). L'utilisateur peut choisir une valeur dans cette liste, et ensuite éditer le texte. En conséquence, pour une combo box, la notion d'élément sélectionné ne s'applique pas.

Avec les combo boxes, il n'y a jamais d'élément sélectionné. A chaque fois que l'utilisateur sélectionne une des valeurs attachées à la zone, cette valeur est placée dans l'élément zéro du tableau. Ensuite, si l'utilisateur modifie le texte, la valeur modifiée est aussi placée dans cet élément zéro.

Exemple

```
 ` Méthode objet Combo Box asCouleurs
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TABLEAU ALPHA(31;asCouleurs;3)
      asCouleurs{1}:="Bleu"
      asCouleurs{2}:="Blanc"
      asCouleurs{3}:="Rouge"
   :(Evenement formulaire=Sur clic)
      Si(asCouleurs{0}#"")
 ` L'objet change automatiquement de valeur
 ` L'utilisation de l'événement Sur clic avec une Combo Box
 ` n'est requise que pour prendre en compte des actions supplémentaires
      Fin de si
   :(Evenement formulaire=Sur données modifiées)
 ` Chercher dans tableau ignore l'élément 0, et donc retourne -1 ou >0
      Si(Chercher dans tableau(asCouleurs;asCouleurs{0})<0)
 ` La valeur saisie n'est pas une des valeurs attachées à l'objet
```

```
    ` Ajouter la valeur à la liste pour la prochaine fois
            AJOUTER A TABLEAU(asCouleurs;asCouleurs{0})
        Sinon
    ` La valeur entrée est une des valeurs attachées à l'objet
        Fin de si
Fin de cas
```

Les tableaux et le langage 4D

Les tableaux sont des variables 4D. Comme toute variable, un tableau a une portée (une aire d'action) et suit les règles du langage 4D, à certaines différences près.

Tableaux locaux, process et interprocess

Vous pouvez créer des tableaux locaux, process ou interprocess, par exemple :

```
TABLEAU ENTIER($aiCodes;100) ` Ceci crée un tableau local de 100 valeurs entières sur 2 octets
TABLEAU ENTIER(aiCodes;100) ` Ceci crée un tableau process de 100 valeurs entières sur 2 octets
TABLEAU ENTIER(◊aiCodes;100) ` Ceci crée un tableau interprocess de 100 valeurs entières sur 2 octets
```

La portée de ces tableaux est identique à celle des autres variables locales, process et interprocess.

Tableaux locaux

Vous déclarez un tableau local lorsque son nom commence par le signe dollar ($).

La portée d'un tableau local est la méthode dans laquelle il est créé. Le tableau est effacé lorsque la méthode est terminée. Des tableaux locaux de même nom peuvent avoir des types différents dans deux méthodes différentes, car ce sont en réalité des variables différentes n'ayant pas la même portée.

Lorsque vous créez un tableau local dans une méthode formulaire ou objet, ou dans une méthode projet appelée comme sous-routine par l'un des deux types de méthodes précédents, le tableau est créé puis effacé à chaque fois que la méthode formulaire ou la méthode objet est utilisée. En d'autres termes, le tableau est créé puis effacé pour chaque événement formulaire. Par conséquent, vous ne pouvez pas utiliser de tableaux locaux dans des formulaires, ni pour l'affichage ni pour l'impression.

Comme dans le cas des variables locales, il est préférable d'utiliser les tableaux locaux à chaque fois que c'est possible. Vous réduisez ainsi la quantité de mémoire nécessaire pour votre application.

Tableaux process

Vous déclarez un tableau process lorsque le nom du tableau débute par une simple lettre.

La portée d'un tableau process est le process dans lequel il a été créé. Le tableau est effacé lorsque le process se termine ou est tué. Un tableau process dispose d'une instance créée automatiquement pour chaque process. Par conséquent, le tableau est du même type pour tous les process. En revanche, son contenu est particulier à chaque process.

Tableaux interprocess

Vous déclarez un tableau interprocess lorsque son nom commence par <> (sous Windows et Mac OS) ou par le signe "diamant" (Mac OS uniquement — Option+v sur un clavier français).

La portée d'un tableau interprocess est la totalité des process pendant la session de travail. Il convient de ne les utiliser que pour partager des données ou transférer des informations entre les process.

Astuce : Quand vous savez d'avance qu'un tableau interprocess sera utilisé par plusieurs process, ce qui peut provoquer des conflits, protégez l'accès à ce tableau par un sémaphore. Pour plus d'informations, reportez-vous à l'exemple de la commande **Semaphore**.

Note : Vous pouvez utiliser des tableaux process et interprocess dans des formulaires pour créer des objets de formulaire tels que des zones de défilement, des listes déroulantes, etc.

Passer un tableau comme paramètre

Vous pouvez passer un tableau en tant que paramètre à une commande 4D ou à une routine d'un plug-in 4D. Cependant, vous ne pouvez pas passer un tableau comme paramètre à une méthode utilisateur. La solution dans ce cas est de passer un pointeur vers le tableau comme paramètre à cette méthode. Pour plus de détails, reportez-vous à la section **Tableaux et pointeurs**.

Affecter un tableau à un autre tableau

Contrairement à ce que vous pouvez faire avec des variables de type Texte ou Chaîne, vous ne pouvez pas affecter un tableau à un autre tableau. Pour copier (affecter) un tableau à un autre, utilisez la fonction **COPIER TABLEAU**.

Tableaux et pointeurs

Vous pouvez passer un tableau comme paramètre à une commande 4D ou à une routine d'un Plug-in 4D. Cependant, vous ne pouvez pas passer un tableau comme paramètre dans une méthode utilisateur. La solution consiste à passer un pointeur vers le tableau comme paramètre de la méthode.

Vous pouvez passer des tableaux interprocess, process ou locaux comme paramètres.

Voici quelques exemples.

- Prenons le cas suivant :

```
Si((0<atNoms)&(atNoms<Taille tableau(atNoms))
  ` Si possible, sélectionner l'élément suivant l'élément sélectionné
   atNoms:=atNoms+1
Fin de si
```

Si vous avez besoin de faire la même chose pour 50 tableaux différents, vous pouvez vous éviter d'écrire 50 fois la même chose, en utilisant la méthode projet suivante:

```
  ` Méthode projet SELECTIONNER ELEMENT SUIVANT
  ` SELECTIONNER ELEMENT SUIVANT ( Pointeur )
  ` SELECTIONNER ELEMENT SUIVANT ( -> Tableau )

C_POINTEUR($1)

Si((0<$1->)&($1-><Taille tableau($1->))
   $1->:=$1->+1 ` Si possible, sélectionner l'élément suivant l'élément sélectionné
Fin de si
```

Ensuite, vous pouvez écrire :

```
SELECTIONNER ELEMENT SUIVANT(->atNoms)
  ` ...
SELECTIONNER ELEMENT SUIVANT(->asCodesPostaux)
  ` ...
SELECTIONNER ELEMENT SUIVANT(->alEnrgsIDs)
  ` ... et ainsi de suite.
```

- La méthode projet suivante retourne la somme de tous les éléments d'un tableau numérique (Entier, Entier long, ou Réel) :

```
  ` Somme Tableau
  ` Somme Tableau ( Pointeur )
  ` Somme Tableau ( -> Tableau )

C_REEL($0)

$0:=0
Boucle($vlElem;1;Taille tableau($1->))
   $0:=$0+$1->{$vlElem}
Fin de boucle
```

Note : Depuis 4D v13, vous pouvez utiliser simplement la fonction **Somme** pour effectuer la somme des éléments d'un tableau numérique.

Ensuite, vous pouvez écrire :

```
$vlSomme:=Somme Tableau(->arSalaires)
  ` ...
$vlSomme:=Somme Tableau(->aiDefectCounts)
  ` ..
$vlSomme:=Somme Tableau(->alPopulations)
```

- La méthode projet qui suit met une majuscule à tous les éléments d'un tableau Alpha ou Texte :

```
  ` MAJUSCULE TABLEAU
  ` MAJUSCULE TABLEAU ( Pointeur )
  ` MAJUSCULE TABLEAU ( -> Tableau )

Boucle($vlElem;1;Taille tableau($1->))
   Si($1->{$vlElem}#"")
      $1->{$vlElem}:=Majusc($1->{$vlElem}≤1≥)+Minusc(Sous chaine($1->{$vlElem};2))
   Fin de si
Fin de boucle
```

Ensuite, vous pouvez écrire :

```
MAJUSCULE TABLEAU(->atSujets)
  ...
MAJUSCULE TABLEAU(->asNomsFamille)
```

La combinaison de tableaux, pointeurs et de boucles telles que **Boucle...Fin de boucle** vous permet d'écrire un grand nombre de petites méthodes projet très utiles pour gérer les tableaux.

Utiliser l'élément zéro d'un tableau

Un tableau a toujours un élément zéro. Même si l'élément zéro n'est pas affiché lorsqu'un tableau est utilisé pour remplir un objet de formulaire, vous pouvez l'utiliser sans réserve(*) dans le langage.

Un exemple possible d'utilisation de l'élément zéro est le cas de la combo box examiné dans la section **Tableaux et objets de formulaire**.

Voici un autre exemple : vous voulez exécuter une action seulement lorsque vous cliquez sur un élément autre que l'élément préalablement sélectionné. Pour cela, vous devez garder la trace de chaque élément sélectionné. Une façon de le faire est d'utiliser une variable process dans laquelle vous conservez le numéro de l'élément sélectionné. Une autre manière consiste à utiliser l'élément zéro du tableau :

```
   ` Méthode objet zone de défilement atNoms
Au cas ou
   :(Evenement formulaire=Sur chargement)
   ` Initialisont le tableau
      TABLEAU TEXTE(atNoms;5)
   ` ...
   ` Initialiser l'élément zéro avec le numéro
   ` de l'élément courant sélectionné sous sa forme alphanumérique
   ` Ici vous commencez sans élément sélectionné
      atNoms{0}:="0"

   :(Evenement formulaire=Sur libération)
   ` Nous n'avons plus besoin du tableau
      EFFACER VARIABLE(atNoms)

   :(Evenement formulaire=Sur clic)
      Si(atNoms#0)
         Si(atNoms#Num(atNoms{0}))
            vtInfo:="Vous avez cliqué sur : "+atNoms{atNoms}+" qui n'était pas précédemment sélectionné."
            atNoms{0}:=Chaine(atNoms)
         Fin de si
      Fin de si
   :(Evenement formulaire=Sur double clic)
      Si(atNoms#0)
         ALERTE("Vous avez double-cliqué sur : "+atNoms{atNoms})
      Fin de si
Fin de cas
```

(*) Il existe une exception : dans les **List Box** de type tableau, l'élément zéro est utilisé en interne pour conserver la valeur précédente d'un élément en cours d'édition. Il n'est donc pas possible de l'utiliser dans ce contexte.

Tableaux à deux dimensions

Chaque commande de déclaration de tableau permet de créer ou de redimensionner des tableaux à une ou à deux dimensions. Exemple :

```
TABLEAU TEXTE(atTopics;100;50) ` Créer un tableau texte composé de 100 lignes de 50 colonnes
```

Les tableaux à deux dimensions sont essentiellement des objets de langage ; vous ne pouvez ni les afficher ni les imprimer.
Dans l'exemple prédédent :

- *atTopics* est un tableau à deux dimensions.
- *atTopics{8} {5}* est le 5e élément (5e colonne...) de la 8e ligne.
- *atTopics{20}* est la 20e ligne et est elle-même un tableau à une dimension.
- **Taille tableau(atTopics)** retourne 100, qui est le nombre de lignes
- **Taille tableau(atTopics{17})** retourne 50, qui est le nombre de colonnes de la 17e ligne

Dans l'exemple suivant, un pointeur vers chaque champ de chaque table de la base est stocké dans un tableau à deux dimensions :

```
C_ENTIER LONG($vlDerniereTable;$vlDernierChamp)
C_ENTIER LONG($vlNumeroChamp)
 ` Créer autant de lignes (vides et sans colonnes) qu'il y a de tables
$vlDerniereTable:=Lire numero derniere table
TABLEAU POINTEUR(<>apChamps;$vlDerniereTable;0) `Tableau 2D avec N lignes et zéro colonnes
 ` Pour chaque table
Boucle($vlTable;1;$vlDerniereTable)
   Si(Est un numero de table valide($vlTable))
      $vlDernierChamp:=Lire numero dernier champ($vlTable)
 ` Donner la valeur des éléments
      $vlNumeroColonne:=0
      Boucle($vlChamp;1;$vlDernierChamp)
         Si(Est un numero de champ valide($vlTable;$vlChamp))
            $vlNumeroColonne:=$vlNumeroColonne+1
 `Insère une colonne dans la ligne de la table en cours
            INSERER DANS TABLEAU(<>apChamps{$vlTable};$vlNumeroColonne;1)
 `Affecte la "celulle" avec le pointeur
            <>apChamps{$vlTable}{$vlNumeroColonne}:=Champ($vlTable;$vlChamp)
         Fin de si
      Fin de boucle
   Fin de si
Fin de boucle
```

Dans la mesure où le tableau à deux dimensions a été initialisé, vous pouvez obtenir ainsi les pointeurs vers les champs d'une table de votre choix :

```
 ` Obtenir les pointeurs vers les champs pour la table affichée à l'écran:
COPIER TABLEAU(◊apChamps{Table(Table du formulaire courant)};$apMesChampsdeTravail)
 ` Initialiser les champs booléens et date
Boucle($vlElem;1;Taille tableau($apMesChampsdeTravail))
   Au cas ou
      :(Type($apMesChampsdeTravail{$vlElem}->)=Is Date)
         $apMesChampsdeTravail{$vlElem}->:=Date du jour
      :(Type($apMesChampsdeTravail{$vlElem}->)=Is Boolean)
         $apMesChampsdeTravail{$vlElem}->:=Vrai
   Fin de cas
Fin de boucle
```

Note : Comme le montre cet exemple, les lignes des tableaux à deux dimensions peuvent être ou non de la même taille, indifféremment.

Tableaux et mémoire

A la différence des données que vous stockez sur disque lorsque vous utilisez des tables ou des enregistrements, un tableau **réside toujours en mémoire dans son intégralité**.

Par exemple, si tous les codes postaux américains étaient saisis dans une table *[Codes Postaux]*, celle-ci contiendrait environ 100 000 enregistrements. De plus, cette table comporterait plusieurs champs : le code postal lui-même ainsi que la ville, le comté et l'état correspondants. Si vous ne sélectionnez que les codes postaux de Californie, 4D crée la sélection d'enregistrements correspondante à l'intérieur de la table *[Codes Postaux]*, et ensuite ne charge les enregistrements que lorsque vous en avez besoin (par exemple, pour les afficher ou les imprimer). En d'autres termes, vous travaillez avec une série ordonnée de valeurs (du même type pour chaque champ) partiellement chargée du disque en mémoire.

Procéder de la même manière avec les tableaux serait laborieux, pour les raisons suivantes :

- Pour maintenir les quatre types d'information (code postal, ville, comté, état), vous auriez besoin de quatre grands tableaux en mémoire.
- Comme un tableau réside en mémoire dans son intégralité, vous seriez obligé de garder tous les codes postaux en mémoire pendant toute la session de travail, même si les données n'étaient pas utilisées en permanence.
- Toujours parce qu'un tableau réside en mémoire dans son intégralité, les quatre tableaux devraient être chargés ou sauvegardés sur le disque à chaque fois que vous démarreriez ou quitteriez l'application, quand bien même les données ne seraient d'aucune utilité pour la session de travail.

Conclusion : Les tableaux ont pour rôle de manipuler une certaine quantité de données pendant une période brève. En contrepartie, comme ils résident en mémoire, ils sont d'une utilisation rapide et facile.

Cependant, dans certaines circonstances, vous pouvez avoir besoin de tableaux contenant des centaines ou des milliers d'éléments. Voici les formules à appliquer pour calculer la quantité de mémoire utilisée pour chaque type de tableau:

Type de Tableau	Calcul de la quantité de mémoire en octets
Booléen	(31+nombre d'éléments)/8
Date	(1+nombre d'éléments) * 6
Alpha	(1+nombre d'éléments) * (somme de la taille de chaque texte)
Entier	(1+nombre d'éléments) * 2
Entier long	(1+nombre d'éléments) * 4
Image	(1+nombre d'éléments) * 4 + somme de la taille de chaque image
Pointeur	(1+nombre d'éléments) * 16
Réel	(1+nombre d'éléments) * 8
Texte	(1+nombre d'éléments) * (somme de la taille de chaque texte)
Deux dimensions	(1+nombre d'éléments) * 12 + somme de la taille de chaque tableau

Notes :

- La taille d'un texte en mémoire se calcule par la formule ((Longueur + 1) * 2)
- Quelques octets supplémentaires sont requis pour le repérage de l'élément, le nombre d'éléments et le tableau lui-même.

Lorsque vous travaillez avec de très grands tableaux, la meilleure façon de gérer d'éventuels problèmes de saturation de la mémoire est d'accompagner la création de tableau d'une méthode projet **APPELER SUR ERREUR**. Exemple :

```
 // Vous allez lancer une opération batch fonctionnant toute la nuit
 // qui requiert la création de grands tableaux. Pour éviter
 // des erreurs en pleine nuit, créez les tableaux au début de
 // l'opération et testez les erreurs au même moment :
gError:=0 ` Initialisation
APPELER SUR ERREUR("GESTION ERREUR") ` Installation de la méthode de gestion d'erreurs
TABLEAU ALPHA(63;asCeTableau;50000) ` Environ 3125 Ko en mode ASCII
TABLEAU REEL(arCetAutreTableau;50000) ` 488 Ko
APPELER SUR ERREUR("") ` Il n'est plus nécessaire d'intercepter les erreurs
Si(gError=0)
 //  Les tableaux ont pu être créés
 //  poursuivons l'opération
Sinon
   ALERTE("Cette opération requiert davantage de mémoire !")
Fin de si
 //  Dans tous les cas, nous n'avons plus besoin des tableaux
EFFACER VARIABLE(asCeTableau)
EFFACER VARIABLE(arCetAutreTableau)
```

La méthode projet ***GESTION ERREUR*** est la suivante :

```
 //  Méthode projet GESTION ERREUR
gError:=Error ` Retourner le code d'erreur
```

AJOUTER A TABLEAU

AJOUTER A TABLEAU (tableau ; valeur)

Paramètre	Type		Description
tableau	Tableau	⇨	Tableau auquel ajouter une valeur
valeur	Expression	⇨	Valeur à ajouter au tableau

Description

La commande **AJOUTER A TABLEAU** ajoute une nouvelle ligne à la fin du *tableau* et lui affecte la valeur passée dans le paramètre *valeur*. En mode interprété, si le *tableau* n'a pas été défini au préalable, la commande le crée et lui attribue un type en fonction de celui de *valeur*.

Cette commande fonctionne avec tous les types de tableaux : chaîne, numérique, booléen, date, pointeur et image.

Le type de *valeur* doit correspondre au type du tableau, sinon l'erreur de syntaxe 54 "Les arguments sont incompatibles" est générée. Les combinaisons suivantes sont toutefois possibles :

- un *tableau* de type chaîne (Texte ou Alpha) accepte toute *valeur* de type Texte ou Alpha.
- un *tableau* de type numérique (Entier, Entier long ou Réel) accepte toute *valeur* de type Entier, Entier long, Numérique ou Heure.

Exemple

Le code suivant :

```
INSERER DANS TABLEAU($montableau;Taille tableau($montableau)+1)
$montableau{Taille tableau($montableau)}:=$mavaleur
```

... peut être remplacé par :

```
AJOUTER A TABLEAU($montableau;$mavaleur)
```

Chercher dans tableau

Chercher dans tableau (tableau ; valeur {; départ}) -> Résultat

Paramètre	Type		Description
tableau	Tableau	➡	Tableau dans lequel effectuer la recherche
valeur	Expression	➡	Valeur de même type à rechercher dans le tableau
départ	Entier long	➡	Elément à partir duquel commencer la recherche
Résultat	Entier long	⮌	Numéro du premier élément trouvé correspondant à valeur

Description

Chercher dans tableau retourne le numéro du premier élément de *tableau* qui correspond à *valeur*.

Chercher dans tableau peut être utilisé avec des tableaux de type Texte, Alpha, Numérique, Date, Pointeur et Booléen. Les paramètres *tableau* et *valeur* doivent être du même type.

Si aucun élément n'est trouvé, **Chercher dans tableau** renvoie -1.

Si *départ* est spécifié, **Chercher dans tableau** commence la recherche à l'élément spécifié par *départ*. Si *départ* n'est pas spécifié, **Chercher dans tableau** commence la recherche à l'élément 1.

Exemple 1

La méthode projet suivante efface tous les éléments vides du tableau alpha ou texte passé en paramètre :

```
  ` Méthode projet NETTOYER TABLEAU
  ` NETTOYER TABLEAU ( Pointeur )
  ` NETTOYER TABLEAU ( -> Tableau Texte ou Alpha )

C_POINTEUR($1)
Repeter
   $vlElem:=Chercher dans tableau($1->;"")
   Si($vlElem>0)
      SUPPRIMER DANS TABLEAU($1->;$vlElem)
   Fin de si
Jusque($vlElem<0)
```

Une fois que cette méthode projet est implémentée dans votre base, vous pouvez écrire, par exemple :

```
TABLEAU TEXTE(TabValeurs;...)
  ` ...
  ` Utiliser le tableau comme vous voulez
  ` ...
  ` Eliminer les éléments chaînes vides
NETTOYER TABLEAU(->TabValeurs)
```

Exemple 2

La méthode projet suivante sélectionne le premier élément d'un tableau dont le pointeur passé comme premier paramètre correspond à la valeur de la variable ou du champ dont le pointeur est passé en second paramètre :

```
  ` Méthode projet SELECTIONNER ELEMENT
  ` SELECTIONNER ELEMENT ( Pointeur ; Pointeur)
  ` SELECTIONNER ELEMENT ( -> Tableau Texte ou Alpha ; -> Champ ou variable de type Texte ou Alpha )

$1->:=Chercher dans tableau($1->;$2->)
Si($1->=-1)
   $1->:=0 ` Si aucun élément n'est trouvé, fixer le tableau à aucun élément sélectionné
Fin de si
```

Une fois que cette méthode projet est implémentée dans la base, vous pouvez écrire, par exemple :

```
  ` Méthode objet du pop-up menu TabTitres
Au cas ou
   :(Evenement formulaire=Sur chargement)
      SELECTIONNER ELEMENT(->TabTitres;->[Personnes]Titre)
Fin de cas
```

Note : Cet exemple utilise l'**élément sélectionné** du tableau. Gardez à l'esprit que l'élément sélectionné ne sera pas significatif si le tableau comporte plus de 32767 éléments (cf. section **Tableaux et objets de formulaire**). Il est dans ce cas nécessaire d'utiliser une variable entier long pour stocker le résultat de **Chercher dans tableau**.

Chercher dans tableau trié

Chercher dans tableau trié (tableau ; valeur ; > ou < {; posDébut {; posFin}}) -> Résultat

Paramètre	Type		Description
tableau	Tableau	⇒	Tableau dans lequel effectuer la recherche
valeur	Expression	⇒	Valeur (de même type que le tableau) à rechercher dans le tableau
> ou <	Opérateur	⇒	> si le tableau est trié par ordre croissant, < s'il est trié par ordre décroissant
posDébut	Entier long	⇐	Si la valeur est trouvée, position de sa première occurrence ; sinon, position où la valeur devrait être insérée
posFin	Entier long	⇐	Si la valeur est trouvée, position de sa dernière occurrence ; sinon, identique à posDébut
Résultat	Booléen	↩	Vrai si au moins un élément du tableau correspond à la valeur recherchée, sinon Faux

Description

La commande **Chercher dans tableau trié** retourne **vrai** si au moins un élément du *tableau* trié correspond à *valeur*, et optionnellement retourne la position du ou des élément(s) trouvé(s). A la différence de **Chercher dans tableau**, **Chercher dans tableau trié** travaille uniquement avec un *tableau* trié et fournit des informations sur la position des occurrences, ce qui permet d'insérer des éléments si nécessaire.

Le *tableau* doit être déjà trié conformément au tri défini par le paramètre *> ou <* (symbole "supérieur à" pour l'ordre croissant et symbole "inférieur à" pour l'ordre décroissant). La commande **Chercher dans tableau trié** tire pleinement parti de ce tri et utilise un algorithme de recherche par dichotomie, qui est généralement plus efficace pour les tableaux de grande taille (pour plus d'informations, vous pouvez consulter la page consacrée à la dichotomie sur Wikipedia). Cependant, si le tableau n'est pas correctement trié, le résultat pourra être incorrect.

La commande ne tiendra pas compte de l'indicateur de tri et se comportera comme une commande **Chercher dans tableau** standard (recherche séquentielle, renvoi de -1 dans *posDébut* et *posFin* si *valeur* n'est pas trouvée) dans les cas suivants :

- si le type du tableau ne peut pas être trié (par exemple tableau de pointeurs),
- si le tableau est de type booléen (non pertinent pour une recherche par dichotomie),
- si la base de données n'est pas en Unicode (mode compatibilité) et si le tableau est un tableau alpha ou texte,
- lorsque vous recherchez, dans un tableau texte, une chaîne incluant un joker (@) au début ou au milieu de la chaîne (la recherche par dichotomie n'est pas utilisable car les éléments correspondants peuvent être non contigus dans le tableau).

Lorsque la commande renvoie **Faux**, la *valeur* retournée dans le paramètre *posDébut* peut être passée à **INSERER DANS TABLEAU** afin de l'insérer à la "bonne" place dans le *tableau*, c'est-à-dire en respectant son tri courant. Cette séquence est plus rapide que l'ajout d'un nouvel élément à la fin du tableau suivi de l'appel à **TRIER TABLEAU** afin de le placer au bon endroit.

La valeur retournée dans *posFin* peut être utilisée conjointement à celle retournée dans *posDébut* afin d'itérer sur chaque élément du tableau correspondant à la *valeur* (via une **Boucle...Fin de boucle**) ou pour trouver le nombre total d'occurrences (comme le ferait la commande **Compter dans tableau**, mais plus rapidement).

Exemple 1

Vous souhaitez insérer une valeur, si nécessaire, tout en conservant le tableau trié :

```
C_ENTIER LONG($pos)
Si(Chercher dans tableau trié($array ;$valeur;>;$pos)
   ALERTE("Trouvé à la position "+Chaine($pos))
Sinon
   INSERER DANS TABLEAU($array ;$pos)
   $array{$pos}:=$valeur
Fin de si
```

Exemple 2

Vous souhaitez trouver le nombre d'occurrences de chaînes débutant par "test" et créer une chaîne qui concatène tous ces éléments :

```
C_ENTIER LONG($posFirst ;$posLast)
C_TEXTE($output)
Si(Chercher dans tableau trié($array ;"test@";>;$posFirst ;$posLast))
   $output:="Trouvé "+Chaine($posLast-$posFirst+1)+" résultats :\n"
Fin de si
Boucle($i ;$posFirst ;$posLast)
   $output:=$output+$array{$i}+"\n"
Fin de boucle
```

Compter dans tableau

Compter dans tableau (tableau ; valeur) -> Résultat

Paramètre	Type		Description
tableau	Tableau	➡	Tableau dans lequel effectuer le comptage
valeur	Expression	➡	Valeur à compter
Résultat	Entier long	⮌	Nombre d'occurrences trouvées

Description

La commande **Compter dans tableau** retourne le nombre d'occurrences de *valeur* dans *tableau*.

Cette commande peut être utilisée avec des tableaux de type Texte, Alpha, Numérique, Date, Pointeur et Booléen. Les paramètres *tableau* et *valeur* doivent être du même type ou d'un type compatible.

Si aucun élément de *tableau* ne correspond à *valeur*, la commande retourne 0.

Exemple

L'exemple suivant permet d'afficher le nombre de lignes sélectionnées dans une list box :

```
  `tBList est le nom d'un tableau de colonne List box
ALERTE(Chaine(Compter dans tableau(tBList;Vrai))+" ligne(s) sélectionnée(s) dans la list box")
```

COPIER TABLEAU

COPIER TABLEAU (source ; destination)

Paramètre	Type		Description
source	Tableau	⇨	Tableau à recopier
destination	Tableau	⇦	Tableau de destination

Description

La commande COPIER TABLEAU crée ou remplace le tableau *destination* avec les mêmes contenu, taille et type que le tableau *source*.

Les tableaux *source* et *destination* peuvent être des tableaux locaux, process ou interprocess. La portée du tableau n'a pas d'importance lors de la duplication des tableaux.

Note : En mode compilé, le tableau *destination* doit être du même type que le tableau *source*.

Exemple

L'exemple suivant remplit un tableau C. Un nouveau tableau, "D", est ensuite créé, contenant les mêmes informations que le tableau C :

```
TOUT SELECTIONNER([Contacts]) ` Sélectionner tous les enregistrements dans la table
SELECTION VERS TABLEAU([Contacts]Société;C) ` Remplir le tableau C avec les données du champ
COPIER TABLEAU(C;D) ` Copier le tableau C dans le tableau D
```

INSERER DANS TABLEAU

INSERER DANS TABLEAU (tableau ; positionDépart {; combien})

Paramètre	Type		Description
tableau	Tableau	⇨	Nom du tableau dans lequel insérer des éléments
positionDépart	Entier long	⇨	Position de départ du ou des élément(s) à insérer
combien	Entier long	⇨	Nombre d'éléments à insérer ou 1 élément si ce paramètre est omis

Description

INSERER DANS TABLEAU insère un ou plusieurs éléments ou "lignes" dans le tableau *tableau*. Les nouveaux éléments sont insérés avant l'élément spécifié par *positionDépart*, et initialisés à la valeur vide du type du tableau. Tous les éléments situés au-delà de *positionDépart* sont décalés vers le bas d'un offset ou de la valeur spécifiée par *combien*.

Si *positionDépart* est supérieur à la taille du tableau, les éléments sont insérés à la fin du tableau.

Le paramètre *combien* représente le nombre de lignes à insérer. Si *combien* n'est pas spécifié, un seul élément est inséré. La taille du tableau est augmentée de *combien*.

Exemple 1

L'exemple suivant insère cinq nouveaux éléments à partir de l'élément 10 :

```
INSERER DANS TABLEAU(unTableau;10;5)
```

Exemple 2

L'exemple suivant ajoute un élément à un tableau :

```
$vlElem:=Taille tableau(unTableau)+1
INSERER DANS TABLEAU(unTableau;$vlElem)
unTableau{$vlElem}:=...
```

LISTE VERS TABLEAU

LISTE VERS TABLEAU (liste ; tableau {; réfEléments})

Paramètre	Type		Description
liste	Chaîne, RefListe	⇒	Nom ou référence de la liste de laquelle copier les éléments du premier niveau
tableau	Tableau	⇐	Tableau dans lequel copier les éléments de la liste
réfEléments	Tableau	⇐	Numéros de référence des éléments de la liste

Description

La commande **LISTE VERS TABLEAU** crée ou remplace le tableau *tableau* avec les éléments du premier niveau de la liste ou de l'énumération *liste*.

Vous pouvez passer dans le paramètre *liste* soit un nom d'énumération (une chaîne) soit une référence de liste hiérarchique (*RefListe*).

Si vous n'avez pas préalablement défini le tableau comme tableau de type Alpha ou Texte, **LISTE VERS TABLEAU** crée un tableau de type Texte par défaut.

Note : En mode compilé, le *tableau* doit avoir été préalablement défini et ne peut pas être retypé.

Le paramètre optionnel *réfEléments* (un tableau de type numérique) retourne les numéros de référence des éléments de la liste.

Vous pouvez utiliser **LISTE VERS TABLEAU** pour construire un tableau basé sur les éléments de premier niveau d'une liste. Cependant, cette commande ne vous donne pas les moyens de travailler avec les éléments des sous-listes. Pour exploiter pleinement les listes hiérarchiques, il est préférable d'utiliser les commandes de listes hiérarchiques, notamment **Charger liste**.

Exemple 1

L'exemple suivant recopie les éléments de l'énumération Régions dans le tableau *tabRégions* :

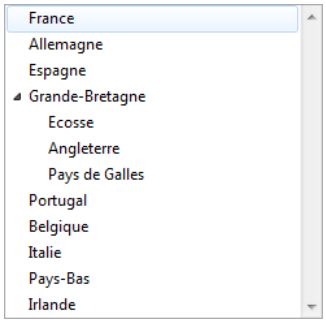
```
LISTE VERS TABLEAU("Régions";tabRégions)
```

Exemple 2

Soit une liste hiérarchique créée de la manière suivante :

```
MyList2:=Nouvelle liste
AJOUTER A LISTE(myList2;"Ecosse";1)
AJOUTER A LISTE(myList2;"Angleterre";2)
AJOUTER A LISTE(myList2;"Pays de Galles";3)
myList1:=Nouvelle liste
AJOUTER A LISTE(myList1;"France";1)
AJOUTER A LISTE(myList1;"Allemagne";2)
AJOUTER A LISTE(myList1;"Espagne";3)
AJOUTER A LISTE(myList1;"Grande-Bretagne";4;MyList2;Vrai)
AJOUTER A LISTE(myList1;"Portugal";5)
AJOUTER A LISTE(myList1;"Belgique";6)
AJOUTER A LISTE(myList1;"Italie";7)
AJOUTER A LISTE(myList1;"Pays-Bas";8)
AJOUTER A LISTE(myList1;"Irlande";9)
```

Cette liste peut être représentée ainsi :



Si vous exécutez l'instruction :

```
LISTE VERS TABLEAU(myList1;$MonTab)
```

...vous obtenez :

```
$MonTab{1}="France"
$MonTab{2}="Allemagne"
$MonTab{3}="Espagne"
$MonTab{4}="Grande-Bretagne"
$MonTab{5}="Portugal"
...
```

SELECTION LIMITEE VERS TABLEAU

SELECTION LIMITEE VERS TABLEAU (début ; fin {; leChamp | laTable ; tableau} {; leChamp | laTable2 ; tableau2 ; ... ; leChamp | laTableN ; tableauN})

Paramètre	Type		Description
début	Entier long	➡	Numéro de l'enregistrement sous-sélectionné à partir duquel commencer la copie des données
fin	Entier long	➡	Numéro de l'enregistrement sous-sélectionné auquel arrêter la copie des données
leChamp | laTable	Champ, Table	➡	Champ à utiliser pour récupérer les données ou Table à utiliser pour récupérer les numéros d'enregistrements
tableau	Tableau	⬅	Tableau recevant les données ou les numéros d'enregistrements

Description

SELECTION LIMITEE VERS TABLEAU crée un ou plusieurs tableau(x) et y copie des données en provenance des champs de la sélection courante ou les numéros des enregistrements de la sélection courante.

A la différence de **SELECTION VERS TABLEAU** qui s'applique à l'intégralité de la sélection courante, **SELECTION LIMITEE VERS TABLEAU** s'applique uniquement à une sous-sélection d'enregistrements, définie par les paramètres *début* et *fin*.

Vous devez passer dans les paramètres *début* et *fin* des numéros d'enregistrements sous-sélectionnés s'inscrivant dans l'intervalle défini par la formule *1 <= début <= fin <= Enregistrements trouves ([...])*.

Si vous passez des numéros correspondant à *1 <= début = fin <= Enregistrements trouves ([...])*, ce sont les champs ou les numéros des enregistrements de la sélection courante répondant à *début = fin* qui seront chargés.

Si vous passez des numéros d'enregistrements incorrects, vous obtiendrez les résultats suivants :

- Si *fin > Enregistrements trouves ([...])*, la commande retourne toutes les valeurs, à partir de l'enregistrement sous-sélectionné spécifié par *début* jusqu'au dernier enregistrement sous-sélectionné.
- Si *début > fin*, la commande ne retourne que les valeurs de l'enregistrement *début*.
- Si les deux paramètres sont incompatibles avec la taille de la sous-sélection, les tableaux sont retournés vides

Comme **SELECTION VERS TABLEAU**, **SELECTION LIMITEE VERS TABLEAU** s'applique à la sélection de la table passée en paramètre. La commande peut réaliser les opérations suivantes :

- Charger les valeurs d'un ou plusieurs champs,
- Charger les numéros des enregistrements, à l'aide de la syntaxe *...;[table];tableau;...*
- Charger des valeurs de champs liés, s'il existe un lien automatique de N vers 1 entre les tables, ou si vous avez préalablement appelé la commande **FIXER LIENS AUTOMATIQUES** pour rendre automatiques les liens manuels N vers 1 (dans les deux cas, les valeurs peuvent être chargées à travers plusieurs niveaux de liens N vers 1 entre les tables).

Chaque tableau est typé en fonction du type de champ.

A noter toutefois que **SELECTION LIMITEE VERS TABLEAU** appliquée à un champ de type Heure créera un tableau de type Heure uniquement si le tableau n'a pas déjà été défini dans un autre type. Par exemple dans le contexte ci-dessous, le tableau *monTab* conservera le type Entier long :

```
TABLEAU ENTIER LONG(monTab;0)
SELECTION LIMITEE VERS TABLEAU([maTable]monChpHeure;monTab)
```

Si vous chargez les numéros des enregistrements, ils sont copiés dans un tableau de type Entier long.

Note : Il est possible d'appeler la commande **SELECTION LIMITEE VERS TABLEAU** avec uniquement les paramètres *début* et *fin*. Cette syntaxe particulière peut être employée pour lancer sur une sélection limitée l'exécution d'une série différée de commandes **SELECTION VERS TABLEAU** utilisant le paramètre * (cf. exemple 4).

4D Server : La commande **SELECTION LIMITEE VERS TABLEAU** est optimisée pour 4D Server. Chaque tableau est créé sur le serveur puis envoyé en totalité sur le poste client.

ATTENTION : SELECTION LIMITEE VERS TABLEAU peut créer des tableaux de taille importante, en fonction de l'intervalle défini par *début* et *fin*, ainsi que du type et de la taille des données à charger. Comme les tableaux résident en mémoire, il peut être utile de tester la taille des tableaux créés après l'exécution de la commande, ou d'utiliser une méthode projet d'interception d'erreurs, installée par la commande **APPELER SUR ERREUR**.

Une fois la commande correctement exécutée, la taille des tableaux résultants est égale à *(fin-début)+1* — sauf si le paramètre *fin* est supérieur au nombre d'enregistrements dans la sélection. Dans ce cas, les tableaux contiennent *(Enregistrements trouves([...])-début)+1* éléments.

Exemple 1

La ligne de code suivante utilise les 50 premiers enregistrements de la sélection courante de la table *[Factures]*. Les valeurs du champ *[Factures]RéfFacture* et du champ lié *[Clients]RéfClient* sont chargées.

```
SELECTION LIMITEE VERS TABLEAU(1;50;[Factures]RéfFacture;tlRéfFacture;[Clients]RéfClient;tlRéfClient)
```

Exemple 2

Les lignes de code suivantes utilisent les 50 derniers enregistrements de la sélection courante de la table *[Factures]*. Les numéros d'enregistrements de la table *[Factures]* ainsi que ceux de la table liée *[Clients]* sont chargés :

```
lTailleSél:=Enregistrements trouves([Factures])
SELECTION LIMITEE VERS TABLEAU(lTailleSél-49;lTailleSél;[Factures];taFactureNum;[Clients];taClientNum)
```

Exemple 3

Les lignes de code suivantes vous permettent de travailler séquentiellement avec des portions de 1000 enregistrements d'une sélection importante qui ne peut pas être chargée dans des tableaux en une seule fois :

```
lMaxPage:=1000
lTailleSél:=Enregistrements trouves([Annuaire])
Boucle($lPage ;1;1+((lTailleSél-1)\lMaxPage))
  ` Charger les valeurs et/ou les numéros d'enregistrements
   SELECTION LIMITEE VERS TABLEAU(1+(lMaxPage*($lPage-1));lMaxPage*$lPage;...;...;...;...;...;...)
  ` Faire quelque chose avec les tableaux
Fin de boucle
```

Exemple 4

Utilisation des 50 premiers enregistrements courants de la table [Factures] pour charger divers tableaux, en exécution différée :

```
    // Instructions différées
SELECTION VERS TABLEAU([Factures]RefFacture;tLRefFac;*)
SELECTION VERS TABLEAU([Factures]Date;tDDateFac;*)
SELECTION VERS TABLEAU([Clients]RefClient;tLRefCli;*)
    // Exécution des instructions différées
SELECTION LIMITEE VERS TABLEAU(1;50)
```

SELECTION VERS TABLEAU

SELECTION VERS TABLEAU {(leChamp | laTable ; tableau {; leChamp ; tableau {; leChamp2 ; tableau2 ; ... ; leChampN ; tableauN}}{; *})}

Paramètre	Type		Description
leChamp | laTable	Champ, Table	⇒	Champ à récupérer dans le tableau ou Table dont les numéros d'enregistrements sont à récupérer dans le tableau
tableau	Tableau	⇐	Tableau recevant les valeurs des champs ou les numéros d'enregistrements
leChamp	Champ	⇒	Champ à récupérer dans le tableau
tableau	Tableau	⇐	Tableau recevant les valeurs du champ
*	Opérateur	⇒	Attente d'exécution

Description

La commande **SELECTION VERS TABLEAU** crée un ou plusieurs tableau(x) et y copie les valeurs des champ(s) ou les numéros d'enregistrement(s) de la sélection courante.

SELECTION VERS TABLEAU s'applique à la sélection courante de la table désignée par le premier paramètre (nom de table ou nom de champ). La commande peut réaliser les opérations suivantes :

- Charger les valeurs d'un ou plusieurs champs,
- Charger les numéros des enregistrements de la table, à l'aide de la syntaxe *[table];tableau*
- Charger des valeurs de champs liés, s'il existe un lien automatique de N vers 1 entre les tables, ou si vous avez préalablement appelé la commande **FIXER LIENS AUTOMATIQUES** pour rendre automatiques les liens manuels N vers 1 (dans les deux cas, les valeurs peuvent être chargées à travers plusieurs niveaux de liens N vers 1 entre les tables).

Chaque tableau est typé en fonction du type de champ.

A noter toutefois que **SELECTION VERS TABLEAU** appliquée à un champ de type Heure créera un tableau de type Heure uniquement si le tableau n'a pas déjà été défini dans un autre type. Par exemple dans le contexte ci-dessous, le tableau *monTab* conservera le type Entier long :

```
TABLEAU ENTIER LONG(monTab;0)
SELECTION VERS TABLEAU([maTable]monChpHeure;monTab)
```

Si vous chargez les numéros des enregistrements, ils sont copiés dans un tableau de type Entier long.

Si vous passez un * en dernier paramètre, 4D n'exécute pas immédiatement la ligne d'instruction correspondante mais la stocke en mémoire ; vous pouvez ainsi empiler plusieurs lignes se terminant par un *. L'ensemble des lignes en attente sera exécuté par une instruction **SELECTION VERS TABLEAU** finale sans paramètre *. A cette fin, la commande peut être appelée sans aucun paramètre. Dans ce cas, les types des tableaux sont vérifiés au moment de l'exécution de la ligne finale (ne contenant pas de *).

A l'image de la commande **CHERCHER**, ce principe vous permet de scinder une instruction complexe en un ensemble de lignes, plus lisibles et plus faciles à maintenir. Il est également possible d'insérer des instructions intermédiaires ou de construire un tableau dans une boucle (cf. exemple 2 de la commande **TABLEAU VERS SELECTION**).

4D Server : La commande **SELECTION VERS TABLEAU** est optimisée pour 4D Server. Chaque tableau est créé sur le serveur puis envoyé en totalité sur le poste client.

ATTENTION : SELECTION VERS TABLEAU peut créer des tableaux de taille importante, en fonction de la taille de la sélection courante, ainsi que du type et de la taille des données à charger. Comme les tableaux résident en mémoire, il peut être utile de tester la taille des tableaux créés après l'exécution de la commande, ou d'utiliser une méthode projet d'appel sur erreur.

Note : Après un appel à **SELECTION VERS TABLEAU**, la sélection courante et l'enregistrement courant ne sont pas modifiés, mais l'enregistrement courant n'est plus chargé. Utilisez la commande **CHARGER ENREGISTREMENT** après un **SELECTION VERS TABLEAU** si vous souhaitez utiliser les valeurs des champs de l'enregistrement courant.

Exemple 1

Dans l'exemple suivant, la table *[Personnes]* dispose d'un lien automatique vers la table *[Sociétés]*. Les deux tableaux *tabNoms* et *tabAdresseSociétés* sont dimensionnés en fonction du nombre d'enregistrements dans la sélection de la table *[Personnes]* et contiennent des informations venant des deux tables :

```
SELECTION VERS TABLEAU([Personnes]Nom;tabNoms;[Sociétés]Adresse;tabAdresseSociétés)
```

Exemple 2

L'exemple ci-dessous retourne les numéros d'enregistrements de la table *[Clients]* dans le tableau *tabNumEnr* et les valeurs du champ *[Clients]Noms* dans le tableau *tabNoms* :

```
SELECTION VERS TABLEAU([Clients];tabNumEnr;[Clients]Noms;tabNoms)
```

Le même exemple peut être écrit :

```
SELECTION VERS TABLEAU([Clients];tabNumEnr;*)
SELECTION VERS TABLEAU([Clients]Noms;tabNoms;*)
SELECTION VERS TABLEAU
```

SUPPRIMER DANS TABLEAU

SUPPRIMER DANS TABLEAU (tableau ; positionDépart {; combien})

Paramètre	Type		Description
tableau	Tableau	⇒	Tableau dans lequel supprimer des lignes
positionDépart	Entier long	⇒	Elément de départ de la suppression
combien	Entier long	⇒	Nombre d'éléments à supprimer ou 1 élément si ce paramètre est omis

Description

La commande **SUPPRIMER DANS TABLEAU** supprime un ou plusieurs élément(s) de *tableau*. Les éléments sont supprimés à partir de l'élément spécifié par *positionDépart*.

Le paramètre *combien* est le nombre d'éléments à supprimer. Si *combien* n'est pas spécifié, un seul élément est supprimé. La taille du tableau est réduite de *combien*.

Exemple 1

L'exemple suivant supprime trois éléments en commençant à l'élément 5 :

```
SUPPRIMER DANS TABLEAU(unTableau;5;3)
```

Exemple 2

L'exemple suivant supprime le dernier élément d'un tableau, s'il existe :

```
$vlElem:=Taille tableau(unTableau)
Si($vlElem>0)
   SUPPRIMER DANS TABLEAU(unTableau;$vlElem)
Fin de si
```

TABLEAU BLOB

TABLEAU BLOB (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU BLOB** crée ou redimensionne un tableau d'éléments de type Blob en mémoire.

Le paramètre *nomTableau* est le nom du tableau.

Le paramètre *taille* est le nombre d'éléments du tableau.

Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, *taille* définit le nombre de lignes et *taille2* le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU BLOB** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à un BLOB vide (**Taille BLOB** = 0).
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type BLOB :

```
TABLEAU BLOB(tabBlob;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type BLOB :

```
TABLEAU BLOB($tabBlob;100;50)
```

Exemple 3

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type BLOB. La variable *$vByteValue* reçoit le 10e octet du BLOB placé dans la 7e colonne et la 5e ligne du tableau BLOB :

```
C_ENTIER($vByteValue)
TABLEAU BLOB($abValues;100;50)
...
$vByteValue:=$abValues{5}{7}{9}
```

TABLEAU BOOLEEN

TABLEAU BOOLEEN (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU BOOLEEN** crée et/ou redimensionne un tableau d'éléments de type *Booléen* en mémoire.

- Le paramètre *nomTableau* est le nom du tableau.
- Le paramètre *taille* est le nombre d'éléments du tableau.
- Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, *taille* spécifie le nombre de lignes et *taille2* spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU BOOLEEN** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à Faux.
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Astuce : Dans certaines circonstances, l'utilisation d'un tableau d'Entiers dans lequel chaque élément différent de zéro signifie "vrai" et chaque élément égal à zéro signifie "faux" est une alternative à l'utilisation d'un tableau de Booléens.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type *Booléen* :

```
TABLEAU BOOLEEN(tabBooléens;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Booléen :

```
TABLEAU BOOLEEN($tabBooléens;100;50)
```

Exemple 3

Cet exemple crée un tableau interprocess de 50 éléments de type Booléen et affecte à chaque élément pair la valeur Faux :

```
TABLEAU BOOLEEN(◊tabBooléens;50)
Boucle($vElem;1;50)
   ◊tabBooléens{$vElem}:=(($vElem%2)=0)
Fin de boucle
```

TABLEAU BOOLEEN SUR ENSEMBLE

TABLEAU BOOLEEN SUR ENSEMBLE (tabBooléen {; ensemble})

Paramètre	Type		Description
tabBooléen	Tableau booléen	⇐	Tableau d'appartenance des enregistrements à l'ensemble
ensemble	Chaîne	⇒	Nom de l'ensemble ou Ensemble UserSet si ce paramètre est omis

Description

La commande **TABLEAU BOOLEEN SUR ENSEMBLE** remplit un tableau de booléens indiquant si chaque enregistrement de la table à laquelle appartient *ensemble* fait ou non partie de l'ensemble.

Les éléments du tableau sont ordonnés en fonction de l'ordre de création des enregistrements dans la table (numéros absolus). Si N est le nombre d'enregistrements de la table, l'élément 0 du tableau correspond à l'enregistrement n° 0, l'élément 1 du tableau correspond à l'enregistrement n° 1, etc.

Chaque élément du tableau est mis à :

- **Vrai** si l'enregistrement correspondant fait partie de l'ensemble,
- **Faux** si l'enregistrement correspondant ne pas partie de l'ensemble.

Attention, le nombre total d'éléments du tableau *tabBooléen* n'est pas significatif. En effet, pour des raisons structurelles, il peut être différent du nombre d'enregistrements effectivement présents dans la table. Les éventuels éléments supplémentaires sont mis à **Faux**.

Si vous ne passez pas le paramètre *ensemble*, la commande utilisera l'ensemble système UserSet du process courant.

TABLEAU DATE

TABLEAU DATE (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU DATE** crée et/ou redimensionne un tableau d'éléments de type *Date* en mémoire.

- Le paramètre *nomTableau* est le nom du tableau.
- Le paramètre *taille* est le nombre d'éléments du tableau.
- Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, *taille* spécifie le nombre de lignes et *taille2* spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU DATE** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à la valeur de date nulle (!00/00/00!).
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type Date :

```
TABLEAU DATE(tabDates;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Date :

```
TABLEAU DATE($tabDates;100;50)
```

Exemple 3

Cet exemple crée un tableau interprocess de 50 éléments de type Date et affecte à chaque élément la date du jour + un nombre de jours égal au numéro de l'élément :

```
TABLEAU DATE(◊tabDates;50)
Boucle($vElem;1;50)
   ◊tabDates{$vElem}:=Date du jour+$vElem
Fin de boucle
```

TABLEAU ENTIER

TABLEAU ENTIER (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU ENTIER** crée et/ou redimensionne un tableau d'éléments de type *Entier* (2 octets) en mémoire.

- Le paramètre *nomTableau* est le nom du tableau.
- Le paramètre *taille* est le nombre d'éléments du tableau.
- Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, *taille* spécifie le nombre de lignes et *taille2* spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU ENTIER** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à 0.
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type *Entier* :

```
TABLEAU ENTIER(tabEntiers;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type *Entier* :

```
TABLEAU ENTIER($tabEntiers;100;50)
```

Exemple 3

Cet exemple crée un tableau interprocess de 50 éléments de type *Entier* et affecte à chaque élément son numéro :

```
TABLEAU ENTIER(◊tabEntiers;50)
Boucle($vElem;1;50)
   ◊tabEntiers{$vElem}:=$vElem
Fin de boucle
```

TABLEAU ENTIER LONG

TABLEAU ENTIER LONG (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	➡	Nom du tableau
taille	Entier long	➡	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	➡	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU ENTIER LONG** crée et/ou redimensionne un tableau d'éléments de type *Entier long* (4 octets) en mémoire.

- Le paramètre *nomTableau* est le nom du tableau.
- Le paramètre *taille* est le nombre d'éléments du tableau.
- Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, *taille* spécifie le nombre de lignes et *taille2* spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU ENTIER LONG** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à 0.
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type *Entier long* :

```
TABLEAU ENTIER LONG(tabEntiersLongs;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type *Entier long* :

```
TABLEAU ENTIER LONG($tabEntiersLongs;100;50)
```

Exemple 3

Cet exemple crée un tableau interprocess de 50 éléments de type *Entier long* et affecte à chaque élément son numéro :

```
TABLEAU ENTIER LONG(◊tabEntiersLongs;50)
Boucle($vElem;1;50)
   ◊tabEntiersLongs{$vElem}:=$vElem
Fin de boucle
```

TABLEAU ENTIER LONG SUR SELECTION

TABLEAU ENTIER LONG SUR SELECTION (laTable ; tabEnrg {; tempo})

Paramètre	Type		Description
laTable	Table	⇒	Table de la sélection courante
tabEnrg	Tableau entier long	⇐	Tableau de numéros d'enregistrements
tempo	Chaîne	⇒	Nom de la sélection temporaire ou Sélection courante si ce paramètre est omis

Description

La commande **TABLEAU ENTIER LONG SUR SELECTION** remplit le tableau *tabEnrg* avec les numéros (absolus) des enregistrements faisant partie de la sélection temporaire *tempo*.

Si vous ne passez pas le paramètre *tempo*, la commande utilise la sélection courante de la table *table*.

Note : L'élément n° 0 du tableau *tabEnrg* est initialisé à -1.

Exemple

Vous voulez récupérer les numéros des enregistrements de la sélection courante :

```
TABLEAU ENTIER LONG($_tRecNum;0) //obligatoire pour le mode compilé
TABLEAU ENTIER LONG SUR SELECTION([Clients];$_tRecNum)
```

TABLEAU HEURE

TABLEAU HEURE (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU HEURE** crée et/ou redimensionne un tableau d'éléments de type Heure en mémoire.

Rappel : Dans 4D, les heures peuvent être traitées en tant que valeurs numériques. Dans les versions de 4D antérieures à la v14, il était nécessaire de combiner un tableau d'entiers longs et un format d'affichage pour gérer un tableau d'heures.

Le paramètre *nomTableau* est le nom du tableau.

Le paramètre *taille* est le nombre d'éléments du tableau.

Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille définit le nombre de lignes et taille2 le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU HEURE** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à la valeur d'heure nulle (00:00:00).
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

A noter que les commandes **SELECTION VERS TABLEAU** et **SELECTION LIMITEE VERS TABLEAU** appliquées à un champ de type Heure créent un tableau de type Heure uniquement si le tableau n'a pas déjà été défini dans un autre type, par exemple en entier long.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type Heure :

```
TABLEAU HEURE(tabHeures;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Heure :

```
TABLEAU HEURE($tabHeures;100;50)
```

Exemple 3

Comme les tableaux d'heures acceptent des valeurs numériques, le code suivant est valide :

```
TABLEAU HEURE($tHValeurs;10)
$CrtHeure:=Heure courante+1
AJOUTER A TABLEAU($tHValeurs;$CrtHeure)
$Found:=Chercher dans tableau($tHValeurs;$CrtHeure)
```

TABLEAU IMAGE

TABLEAU IMAGE (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU IMAGE** crée et/ou redimensionne un tableau d'éléments de type *Image* en mémoire.

- Le paramètre *nomTableau* est le nom du tableau.
- Le paramètre *taille* est le nombre d'éléments du tableau.
- Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, *taille* spécifie le nombre de lignes et *taille2* spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU IMAGE** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à une image vide (ce qui signifie que la fonction **Taille image** appliquée à l'un de ces éléments retourne 0).
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type *Image* :

```
TABLEAU IMAGE(tabImages;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type *Image* :

```
TABLEAU IMAGE($tabImages;100;50)
```

Exemple 3

Cet exemple crée un tableau interprocess d'éléments de type *Image*. La taille du tableau est égale au nombre de ressources *'PICT'* dont le nom commence par *"Utilisateur Intf/"* disponibles dans la base. Chaque image est chargée dans un élément du tableau :

```
LISTE RESSOURCES("PICT";$aiResIDs;$asResNoms)
TABLEAU IMAGE(◊tabImages;Taille tableau($aiResIDs))
$vlPictElem:=0
Boucle($vlElem;1;Taille tableau(◊tabImages))
   Si($asResNoms{$vlElem}="Utilisateur Intf/@")
      $vlPictElem:=$vlPictElem+1
      LIRE RESSOURCE IMAGE("PICT";$aiResIDs{$vlElem};$vgImage)
      ◊tabImages{$vlPictElem}:=$vgImage
   Fin de si
Fin de boucle
TABLEAU IMAGE(◊tabImages;$vlPictElem)
```

TABLEAU MULTI TRI

TABLEAU MULTI TRI (tableau {; sensDuTri}{; tableau2 ; sensDuTri2 ; ... ; tableauN ; sensDuTriN})

Paramètre	Type		Description
tableau	Tableau	⇒	Tableau(x) à trier
sensDuTri	Opérateur	⇒	">" pour effectuer un tri croissant ou "<" pour effectuer un tri décroissant Si omis = pas de tri

TABLEAU MULTI TRI (tabPointeurs ; tabTris)

Paramètre	Type		Description
tabPointeurs	Tableau pointeur	⇒	Tableau de pointeurs de tableaux
tabTris	Tableau entier long	⇒	Tableau d'ordres de tri (1 = tri par ordre croissant, -1 = tri par ordre décroissant, 0 = synchronisation avec des tris précédents)

Description

La commande **TABLEAU MULTI TRI** vous permet d'effectuer un tri multi-critères sur un ensemble de tableaux.

Cette commande admet deux syntaxes différentes.

- **Première syntaxe : TABLEAU MULTI TRI (tableau{; sensDuTri}{; tableau2; sensDuTri2; ...; tableauN; sensDuTriN})**

Cette syntaxe est la plus simple, elle permet de passer directement les noms des tableaux synchronisés auxquels vous souhaitez appliquer un tri multi-critères.

Vous pouvez passer un nombre illimité de couples (*tableau;*> *ou* <) et/ou de tableaux seuls. Tous les tableaux passés en paramètres sont triés de manière synchronisée.

Vous pouvez passer des tableaux de tout type, à l'exception des tableaux de pointeurs et d'images. Vous pouvez trier un élément d'un tableau à deux dimensions (c'est-à-dire *t2DTableau{$vlCetElément}*), mais vous ne pouvez pas trier le tableau 2D lui-même (c'est-à-dire *t2DTableau*).

Pour utiliser le contenu d'un tableau comme critère de tri, passez le paramètre *sensDuTri*. La valeur du paramètre (> ou <) définit l'ordre (croissant ou décroissant) dans lequel le tableau sera trié. Si le paramètre *sensDuTri* est omis, le contenu du tableau n'est pas utilisé comme critère de tri.

Note : Attention, au moins un critère de tri doit être passé pour que la commande fonctionne. Si aucun critère de tri n'est défini, une erreur est générée.

Les niveaux de tris sont déterminés par l'ordre dans lequel les tableaux sont passés à la commande : la position d'un tableau avec critère dans la syntaxe détermine son niveau de tri.

- **Seconde syntaxe : TABLEAU MULTI TRI (tabPointeurs; tabTris)**

Cette syntaxe, plus complexe, est précieuse pour les développements génériques (par exemple, vous pouvez créer une méthode générique de tri des tableaux de tout type, ou encore générer l'équivalent d'un **TRIER TABLEAU** générique).

La paramètre *tabPointeurs* contient le nom d'un tableau de pointeurs de tableaux ; chaque élément de ce tableau est un pointeur désignant un tableau à trier. Les tris seront effectués dans l'ordre des pointeurs de tableaux défini par *tabPointeurs*. **Attention**, tous les tableaux pointés par *tabPointeurs* doivent avoir le même nombre d'éléments.

Note : *tabPointeurs* peut être un tableau de pointeurs local ($nomTabPtr), process (nomTabPtr) ou interprocess (<>nomTabPtr). En revanche, les éléments de ce tableau doivent pointer sur des tableaux process ou interprocess uniquement.

La paramètre *tabTri*s contient le nom d'un tableau dont chaque élément indique l'ordre de tri (-1, 0 ou 1) de l'élément du tableau de pointeurs correspondant :
-1 = Tri par ordre décroissant.
0 = Le tableau n'est pas utilisé comme critère de tri mais doit être trié en fonction des autres tris.
1 = Tri par ordre croissant.

Note : Vous ne pouvez pas trier de tableaux de type Pointeur ou Image. Vous pouvez trier un élément d'un tableau à deux dimensions (c'est-à-dire t2DTableau{$vlCetElément}), mais vous ne pouvez pas trier le tableau 2D lui-même (c'est-à-dire t2DTableau).

A chaque élément du tableau *tabPointeurs* doit correspondre un élément du tableau *tabTri*s. Les deux tableaux doivent donc avoir exactement le même nombre d'éléments.

Exemple 1

L'exemple suivant utilise la première syntaxe : il crée quatre tableaux et les trie par ville (ordre croissant) puis par salaire (ordre décroissant), les deux derniers tableaux *tab_Noms* et *tab_NumTel* étant synchronisés en fonction des critères de tri précédents :

```
TOUT SELECTIONNER([Employés])
SELECTION VERS TABLEAU([Employés]Ville;tab_Villes;[Employés]Salaire;tab_Salaire;[Employés]Nom;
tab_Noms;[Employés]NumTel;tab_NumTel)
TABLEAU MULTI TRI(tab_Villes;>;tab_Salaire;<;tab_Noms;tab_NumTel)
```

Si vous souhaitez que le tableau des noms soit utilisé comme troisième critère de tri, il vous suffit d'ajouter > ou < derrière le paramètre *tab_Noms*.
A noter que la syntaxe :

```
TABLEAU MULTI TRI(tab_Villes;>;tab_Salaire;tab_Noms;tab_NumTel)
```

équivaut strictement à :

```
TRIER TABLEAU(tab_Villes;tab_Salaire;tab_Noms;tab_NumTel;>)
```

Exemple 2

L'exemple suivant utilise la seconde syntaxe : il crée quatre tableaux et les trie par ville (ordre croissant) et société (ordre décroissant), les deux derniers

tableaux tab_Noms et tab_NumTel étant synchronisés en fonction des critères de tri précédents :

```
TOUT SELECTIONNER([Employés])
SELECTION VERS TABLEAU([Employés]Ville;tab_Villes;[Employés]Société;tab_Société;[Employés]Nom;
tab_Noms;[Employés]NumTel;tab_NumTel)
TABLEAU POINTEUR(tab_Pointeurs;4)
TABLEAU ENTIER LONG(tab_Tris;4)
tab_Pointeurs{1}:=->tab_Villes
tab_Tris{1}:=1
tab_Pointeurs{2}:=->tab_Société
tab_Tris{2}:=-1
tab_Pointeurs{3}:=->tab_Noms
tab_Tris{3}:=0
tab_Pointeurs{4}:=->tab_NumTel
tab_Tris{4}:=0
TABLEAU MULTI TRI(tab_Pointeurs;tab_Tris)
```

Si vous souhaitez que le tableau des noms soit utilisé comme troisième critère de tri, il vous suffit d'assigner la valeur 1 à l'élément tab_Tris{3}. Ou bien, si vous souhaitez que les tableaux soient triés uniquement sur le critère des villes, assignez la valeur 0 aux éléments tab_Tris{2}, tab_Tris{3} et tab_Tris{4}. De cette manière, vous obtenez un résultat identique à **TRIER TABLEAU(tab_Villes;tab_Société;tab_Noms;tab_NumTel;>)**.

TABLEAU OBJET

TABLEAU OBJET (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU OBJET** crée et/ou redimensionne un tableau d'éléments de type Objet de langage en mémoire.

Le paramètre *nomTableau* est le nom du tableau. Vous pouvez utiliser tout nom conforme aux conventions de 4D.

Le paramètre *taille* est le nombre d'éléments du tableau.

Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, *taille* définit le nombre de lignes et *taille2* le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes du thème "Tableaux", lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU OBJET** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont indéfinis. Vous pouvez tester si un élément est défini à l'aide de la commande **OB Est defini**.
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemple 1

Création d'un tableau process de 100 éléments de type Objet :

```
TABLEAU OBJET(tabObjets;100)
```

Exemple 2

Création d'un tableau local de 100 lignes contenant chacune 50 éléments de type Objet :

```
TABLEAU OBJET($tabObjets;100;50)
```

Exemple 3

Création et remplissage d'un tableau local d'objets :

```
C_OBJET($Children;$ref_richard;$ref_susan;$ref_james)
TABLEAU OBJET($arrayChildren;0)
OB FIXER($ref_richard;"nom";"Richard";"age";7)
AJOUTER A TABLEAU($arrayChildren;$ref_richard)
OB FIXER($ref_susan;"nom";"Susan";"age";4)
AJOUTER A TABLEAU($arrayChildren;$ref_susan)
OB FIXER($ref_james;"nom";"James";"age";3)
AJOUTER A TABLEAU($arrayChildren;$ref_james)
    //$arrayChildren{1} -> {"nom":"Richard","age":7}
    //$arrayChildren{2} ->  {"nom":"Susan","age":4}
    //$arrayChildren{3}  -> {"nom":"James","age":3}
```

TABLEAU POINTEUR

TABLEAU POINTEUR (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU POINTEUR** crée ou redimensionne un tableau d'éléments de type *Pointeur* en mémoire.

- Le paramètre *nomTableau* est le nom du tableau.
- Le paramètre *taille* est le nombre d'éléments du tableau.
- Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, *taille* spécifie le nombre de lignes et *taille2* spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU POINTEUR** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à un pointeur nul (ce qui signifie que la fonction **Nil** appliquée à l'un de ces éléments retourne Vrai).
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type *Pointeur* :

```
TABLEAU POINTEUR(tabPointeurs;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type *Pointeur* :

```
TABLEAU POINTEUR($tabPointeurs;100;50)
```

Exemple 3

Cet exemple crée un tableau interprocess d'éléments de type *Pointeur* dont la taille est égale au nombre de tables dans la base et remplit chaque élément pointant vers la table dont le numéro est le même que celui de l'élément. Dans la cas d'une table supprimée, la ligne retournera Nil.

```
TABLEAU POINTEUR(<>tabPointeurs;Lire numero derniere table)
Boucle($vElem;Taille tableau(<>tabPointeurs);1;-1)
   Si(Est un numero de table valide($vElem))
      <>tabPointeurs{$vElem}:=Table($vElem)
   Fin de si
Fin de boucle
```

TABLEAU REEL

TABLEAU REEL (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU REEL** crée et/ou redimensionne un tableau d'éléments de type Réel en mémoire.

- Le paramètre *nomTableau* est le nom du tableau.
- Le paramètre *taille* est le nombre d'éléments du tableau.
- Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et *taille2* spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU REEL** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à 0.
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type *Réel* :

```
TABLEAU REEL(tabRéel;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Réel :

```
TABLEAU REEL($tabRéel;100;50)
```

Exemple 3

Cet exemple crée un tableau interprocess de 50 éléments de type *Réel* et affecte à chaque élément son numéro :

```
TABLEAU REEL(◊tabRéel;50)
Boucle($vElem;1;50)
   ◊tabRéel{$vElem}:=$vElem
Fin de boucle
```

TABLEAU TEXTE

TABLEAU TEXTE (nomTableau ; taille {; taille2})

Paramètre	Type		Description
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Description

La commande **TABLEAU TEXTE** crée et/ou redimensionne un tableau d'éléments de type *Texte* en mémoire.

- Le paramètre *nomTableau* est le nom du tableau.
- Le paramètre *taille* est le nombre d'éléments du tableau.
- Le paramètre *taille2* est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, *taille* spécifie le nombre de lignes et *taille2* spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande **TABLEAU TEXTE** à un tableau existant :

- Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à "" (chaîne vide).
- Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemple 1

Cet exemple crée un tableau process contenant 100 éléments de type *Texte* :

```
TABLEAU TEXTE(tabTexte;100)
```

Exemple 2

Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type *Texte* :

```
TABLEAU TEXTE($tabEntiersLongs;100;50)
```

Exemple 3

Cet exemple crée un tableau interprocess de 50 éléments de type *Texte* et affecte à chaque élément la valeur "Elément No" suivie du numéro de l'élément :

```
TABLEAU TEXTE(◊tabTexte;50)
Boucle($vElem;1;50)
   ◊tabTexte{$vElem}:="Elément n°"+Chaine($vElem)
Fin de boucle
```

TABLEAU VERS LISTE

TABLEAU VERS LISTE (tableau ; liste {; réfEléments})

Paramètre	Type		Description
tableau	Tableau	⇒	Tableau duquel copier les éléments
liste	Chaîne, RefListe	⇒	Nom ou référence de la liste dans laquelle copier les éléments du tableau
réfEléments	Tableau	⇒	Tableau numérique des numéros de référence des éléments

Description

La commande **TABLEAU VERS LISTE** crée ou remplace la liste hiérarchique ou l'énumération *liste* en utilisant les éléments du tableau *tableau*.

Vous pouvez passer dans le paramètre *liste* soit un nom d'énumération (une chaîne) soit une référence de liste hiérarchique (*RefListe*). Dans ce deuxième cas, la liste doit déjà avoir été créée (par exemple via la commande **Nouvelle liste**) pour que la commande fonctionne.

Le paramètre optionnel *réfEléments*, s'il est passé, doit être un tableau de type numérique synchronisé avec le tableau *tableau*. Chaque élément de ce tableau indique le numéro de référence de l'élément de la liste correspondant dans *tableau*. Si ce paramètre est omis, 4D affecte automatiquement aux éléments de la liste les numéros de référence 1, 2... N.

Note de compatibilité : La commande **TABLEAU VERS LISTE** doit être utilisée avec précaution du fait des limitations suivantes :

- cette commande permet de définir seulement les éléments du premier niveau de la liste.
- lorsque vous l'utilisez avec une énumération, cette commande modifie la structure de l'application (les énumérations sont stockées dans le fichier de structure), les modifications effectuées en local seront donc perdues lors de mise à jour du fichier de structure en production.
- cette commande ne peut pas être utilisée dans un composant avec une énumération car les composants sont chargés avec la structure en lecture seulement.

Vous pouvez utiliser **TABLEAU VERS LISTE** pour construire une liste basée sur les éléments d'un tableau. Cependant, pour vous affranchir de ces contraintes et exploiter pleinement les listes de valeurs, il est conseillé d'utiliser les commandes du thème **Listes hiérarchiques**.

Exemple

L'exemple suivant copie le tableau *tabRégions* dans l'énumération "Régions" :

```
TABLEAU VERS LISTE(tabRégions;"Régions")
```

Exemple

Vous souhaitez placer les valeurs distinctes d'un champ dans une liste, par exemple pour créer un pop up menu hiérarchique. Vous pouvez écrire :

```
TOUT SELECTIONNER([Company])
VALEURS DISTINCTES([Company]country;$tabPays)
listePays:=Nouvelle liste
TABLEAU VERS LISTE($tabPays;listePays)
```

Gestion des erreurs

La commande **TABLEAU VERS LISTE** génère l'erreur *-9957* lorsqu'elle est appliquée à une énumération en cours de modification en mode Développement. Vous pouvez intercepter cette erreur à l'aide d'une méthode projet de gestion des erreurs installée par la commande **APPELER SUR ERREUR**.

TABLEAU VERS SELECTION

TABLEAU VERS SELECTION {(tableau ; leChamp {; tableau2 ; leChamp2 ; ... ; tableauN ; leChampN}{; *})}

Paramètre	Type		Description
tableau	Tableau	➡	Tableau à copier dans la sélection
leChamp	Champ	⬅	Champ recevant les valeurs du tableau
*	Opérateur	➡	Attente d'exécution

Description

La commande **TABLEAU VERS SELECTION** copie un ou plusieurs tableau(x) vers une sélection d'enregistrements. Tous les champs listés doivent appartenir à la même table.

Si une sélection existe au moment de l'appel, les éléments du tableau sont copiés dans les enregistrements en fonction de l'ordre du tableau et de l'ordre des enregistrements. Si le nombre d'éléments du tableau est supérieur au nombre d'enregistrements de la sélection courante, de nouveaux enregistrements sont créés. Les enregistrements, qu'ils soient nouveaux ou existants, sont automatiquement sauvegardés.

Note : Comme elle permet de créer de nouveaux enregistrements, cette commande ne tient pas compte de l'état lecture seulement éventuel de la table (cf. **Verrouillage d'enregistrements**).

Tous les tableaux doivent avoir le même nombre d'éléments. Si des tableaux ont des tailles différentes, une erreur de syntaxe est générée.

Cette commande effectue l'opération inverse de **SELECTION VERS TABLEAU**. Cependant, **TABLEAU VERS SELECTION** ne permet pas d'utiliser de champs en provenance de tables différentes ni de tables liées, même si un lien automatique existe.

Si vous passez un * en dernier paramètre, 4D n'exécute pas immédiatement la ligne d'instruction correspondante mais la stocke en mémoire ; vous pouvez ainsi empiler plusieurs lignes se terminant par un *. L'ensemble des lignes en attente sera exécuté par une instruction **TABLEAU VERS SELECTION** finale sans paramètre *. A cette fin, la commande peut être appelée sans aucun paramètre.
A l'image de la commande **CHERCHER**, ce principe vous permet de scinder une instruction complexe en un ensemble de lignes, plus lisibles et plus faciles à maintenir. Il est également possible d'insérer des instructions intermédiaires.

ATTENTION : Comme **TABLEAU VERS SELECTION** remplace les informations éventuellement présentes dans les enregistrements existants, cette commande doit être utilisée avec prudence. Si un enregistrement est verrouillé par un autre process pendant l'exécution de la commande **TABLEAU VERS SELECTION**, il n'est pas modifié. Tous les enregistrements verrouillés sont placés dans l'ensemble **LockedSet**. Après l'exécution de **TABLEAU VERS SELECTION**, vous pouvez tester si l'ensemble **LockedSet** contient des enregistrements qui étaient verrouillés.

Note : Cette commande ne tient pas compte du statut lecture seule/lecture écriture de la table du champ.

4D Server : Cette commande est optimisée pour 4D Server. Le tableau est envoyé au serveur depuis le poste client. Les enregistrements sont modifiés ou créés sur le serveur. Comme une telle requête est gérée de façon synchrone, le poste client doit attendre que l'opération se soit correctement déroulée. Dans les environnements multi-utilisateurs et multi-process, aucun enregistrement verrouillé ne sera réécrit.

Exemple 1

Dans l'exemple suivant, les deux tableaux *tabNoms* et *tabSociétés* écrivent des données dans la table *[Personnes]*. Les valeurs du tableau *tabNoms* sont placées dans le champ *[Personnes]Nom* et les valeurs du tableau *tabSociétés* sont placées dans le champ *[Personnes]Société* :

```
TABLEAU VERS SELECTION(tabNoms;[Personnes]Nom;tabSociétés;[Personnes]Société)
```

Exemple 2

Cet exemple permet de dupliquer une sélection d'enregistrements :

```
TABLEAU TEXTE(a1;0)
TABLEAU TEXTE(a2;0)
TABLEAU TEXTE(a3;0)
TABLEAU TEXTE(a4;0)

TOUT SELECTIONNER([Table_1])
Boucle($i;1;Lire numero dernier champ(1))
   $p:=Pointeur vers("a"+Chaine($i))
   SELECTION VERS TABLEAU(Champ(1;$i)->;$p->;*)  // Construction différée des tableaux
Fin de boucle
SELECTION VERS TABLEAU //Exécution des instructions

REDUIRE SELECTION([Table_1];0)
Boucle($i;1;Lire numero dernier champ(1))
   $p:=Pointeur vers("a"+Chaine($i))
   TABLEAU VERS SELECTION($p->;Champ(1;$i)->;*) // Construction de la sélection
Fin de boucle
TABLEAU VERS SELECTION //Exécution des instructions
```

Taille tableau

Taille tableau (tableau) -> Résultat

Paramètre	Type		Description
tableau	Tableau	➡	Tableau dont vous désirez connaître la taille
Résultat	Entier long	⮌	Nombre d'éléments dans le tableau

Description

Taille tableau retourne le nombre d'éléments de *tableau*.

Exemple 1

L'exemple suivant retourne la taille du tableau *monTableau* :

```
vTaille:=Taille tableau(monTableau) ` vTaille reçoit la taille de monTableau
```

Exemple 2

L'exemple suivant retourne le nombre de lignes d'un tableau à deux dimensions :

```
vlLignes:=Taille tableau(t2DTableau) ` vlLignes reçoit la taille de t2DTableau
```

Exemple 3

L'exemple suivant retourne le nombre de colonnes d'une ligne d'un tableau à deux dimensions :

```
vlColonnes:=Taille tableau(t2DTableau{10}) ` vlColonnes reçoit la taille de t2DTableau{10}
```

TEXTE VERS TABLEAU

TEXTE VERS TABLEAU (varTexte ; tabTexte ; largeur ; nomPolice ; taillePolice {; stylePolice {; *}})

Paramètre	Type		Description
varTexte	Texte	⇒	Texte original à découper
tabTexte	Tableau texte	⇐	Tableau contenant le texte découpé en mots ou lignes
largeur	Entier long	⇒	Largeur maximale de la chaîne (en pixels)
nomPolice	Texte	⇒	Nom de police
taillePolice	Entier long	⇒	Taille de police
stylePolice	Entier long	⇒	Style de police
*	Opérateur	⇒	Si passé = interpréter le texte en multistyle

Description

La commande **TEXTE VERS TABLEAU** permet de transformer une variable texte en tableau texte. Le texte d'origine (stylé ou non) est découpé et chaque morceau devient un élément du tableau *tabTexte* qui est retourné par la commande. Cette commande peut être utilisée par exemple pour remplir des pages ou des colonnes de texte de taille fixe.

Le découpage du texte original est effectué en "mots" à partir d'une taille de ligne définie par les paramètres de la commande et tenant compte des styles utilisés.

Passez dans *varTexte* le texte à découper en éléments de tableaux. Ce texte peut être multistyle ou non. Certains paramètres seront ignorés si le texte est multistyle.

Passez dans *tabTexte* le nom du tableau qui sera rempli par le texte découpé.

Passez dans *largeur* une taille en pixels indiquant la longueur maximum de ligne à mesurer pour découper le texte. Pour l'ensemble du texte, la commande évaluera le nombre maximum de mots pouvant "tenir" dans cette largeur en fonction des attributs graphiques du texte (police, style).

- Si le texte est multistyle, les styles du texte original seront pris en compte et les paramètres suivants seront ignorés s'ils sont passés. Dans ce cas, les lignes de texte dans le tableau résultant conserveront leurs styles (afin de pouvoir être imprimées une par une via une variable texte ou alpha par exemple).
- Si le texte est brut (sans styles), vous devez passer tous les paramètres afin que la commande puisse calculer la longueur des lignes.

Chaque élément du tableau doit contenir au moins un mot. Si la *largeur* passée est trop faible pour que la règle de découpage soit entièrement respectée, le tableau sera rempli de la façon la plus proche possible des paramètres et la variable OK prendra la valeur 0. Par exemple, si vous passez une largeur de 3 pixels, il est probable que la taille de la plupart des mots sera au-delà de cette longueur. Dans ce cas, la variable OK prendra la valeur 0.
Ce principe implique également que la taille théorique maximale du tableau retourné est égale au nombre de mots présents dans *varTexte*.

Passez dans *nomPolice* et *taillePolice* le nom et la taille de la police de caractères avec laquelle *varTexte* doit être évalué par la commande pour effectuer le découpage. Ces paramètres sont obligatoires dans le cas d'un texte brut.

Passez dans *stylePolice* une ou une combinaison de constante(s) du thème **Styles de caractères** :

Constante	Type	Valeur
Gras	Entier long	1
Italique	Entier long	2
Normal	Entier long	0
Souligné	Entier long	4

Ce paramètre est optionnel ; s'il est omis, le style Normal est utilisé.

Le paramètre optionnel *, s'il est passé, permet de forcer la prise en compte des paramètres *nomPolice*, *taillePolice* et/ou *stylePolice* pour les textes multistyles lorsque ces paramètres ne sont pas définis dans le texte d'origine. S'ils sont définis dans le texte, les paramètres passés à la commande sont ignorés dans tous les cas.

Exemple 1

Nous souhaitons découper un texte multistyle en lignes d'une taille maximale de 200 pixels :

```
TEXTE VERS TABLEAU(leTexte;leTabTexte;200;"Arial";20;Normal;*)
 // les attributs Arial, 20, Normal ne seront pris en compte que s'ils ne sont pas définis dans le texte
```

Exemple 2

Nous souhaitons découper un texte brut en lignes d'une taille maximale de 350 pixels en police Bodoni gras 14. Comme la commande ne fonctionne pas correctement si la police n'est pas disponible, il est utile de vérifier sa présence :

```
TABLEAU TEXTE($FontList;0)
LISTE DES POLICES($FontList)
$Font:="Bodoni"
$p:=Chercher dans tableau($FontList;$Font)
Si($p>0)
   TEXTE VERS TABLEAU(leTexte;leTabTexte;350;"Bodoni";14;Gras)
Sinon
 // utiliser une autre police.
Fin de si
```

Exemple 3

Un texte multistyle doit être imprimé sans style dans la police Arial normal 12 avec une largeur maximale de 600 pixels :

```
 // on transforme le texte multistyle en texte brut
$RawText:=OBJET Lire texte brut(vText)
 // on remplit le tableau
TEXTE VERS TABLEAU($RawText;tabTexte;600;"Arial";12)
```

Exemple 4

Vous devez imprimer dans une zone de 400 pixels de large un texte d'un maximum de 80 lignes et ce, avec la plus grande taille de police possible (ne devant pas dépasser 24 points). Vous pouvez écrire :

```
TABLEAU TEXTE(tabTexte;0)
$Taille:=24
Repeter
   TEXTE VERS TABLEAU($RawText;tabTexte;400;"Arial";$Taille)
   $Taille:=$Taille-1
   $n:=Taille tableau(tabTexte)
Jusque($n<=80)
```

TRIER TABLEAU

TRIER TABLEAU (tableau {; tableau2 ; ... ; tableauN}{; > ou <})

Paramètre	Type		Description
tableau	Tableau	➡	Tableau(x) à trier
> ou <	Opérateur	➡	">" pour effectuer un tri par ordre croissant ou "<" pour effectuer un tri par ordre décroissant (tri croissant si omis)

Description

La commande **TRIER TABLEAU** trie un ou plusieurs tableau(x) par ordre croissant ou décroissant.

Note : Vous ne pouvez pas trier de tableaux de type *Pointeur* ou *Image*. Vous pouvez trier un élément d'un tableau à deux dimensions (c'est-à-dire *t2DTableau{$vlCetElément}*), mais vous ne pouvez pas trier le tableau 2D lui-même (c'est-à-dire *t2DTableau*).

Le second paramètre spécifie l'ordre du tri : croissant ou décroissant. Si ce paramètre est égal au symbole “supérieur à” (>), l'ordre du tri est croissant. S'il est égal au symbole “inférieur à” (<), l'ordre du tri est décroissant. S'il est omis, l'ordre du tri est croissant.

Si plus d'un tableau est spécifié, les tableaux sont triés en fonction de l'ordre défini pour le premier tableau (les tris multi-niveaux ne sont pas possibles dans ce cas). Utilisez plutôt la commande **TABLEAU MULTI TRI** si vous souhaitez effectuer des tris de tableaux synchronisés.

Exemple 1

L'exemple suivant crée deux tableaux et les trie en fonction du nom de la société :

```
TOUT SELECTIONNER([Personnes])
SELECTION VERS TABLEAU([Personnes]Noms;tabNoms;[Personnes]Sociétés;tabSociétés)
TRIER TABLEAU(tabSociétés;tabNoms;>)
```

Cependant, comme **TRIER TABLEAU** n'effectue pas de tris multi-niveaux, les noms des personnes apparaîtront en désordre à l'intérieur de chaque société. Pour que les noms des personnes soient triés pour chaque société, vous devrez plutôt écrire :

```
TOUT SELECTIONNER([Personnes])
TRIER([Personnes];[Personnes]Sociétés;>;[Personnes]Noms;>)
SELECTION VERS TABLEAU([Personnes]Noms;tabNoms;[Personnes]Sociétés;tabSociétés)
```

Exemple 2

Vous affichez les noms d'une table [Personnes] dans une fenêtre flottante. Cette liste de noms peut être triée de A vers Z ou de Z vers A en fonction du bouton sur lequel vous cliquez, dans la fenêtre. Comme il se peut que certaines personnes portent le même nom, vous avez également créé un champ *[Personnes]Numéro ID* qui est un champ indexé unique. Lorsque vous cliquez sur un nom dans la liste, vous voulez récupérer l'enregistrement correspondant. En utilisant un tableau synchronisé et caché des numéros d'ID, vous êtes certain d'accéder à l'enregistrement correspondant au nom sélectionné :

```
  ` Méthode objet du tableau tabNoms
Au cas ou
   :(Evenement formulaire=Sur chargement)
      TOUT SELECTIONNER([Personnes])
      SELECTION VERS TABLEAU([Personnes]Noms;tabNoms;[Personnes]Numéro ID;tabIDs)
      TRIER TABLEAU(tabNoms;tabIDs;>)
   :(Evenement formulaire=Sur libération)
      EFFACER VARIABLE(tabNoms)
      EFFACER VARIABLE(tabIDs)
   :(Evenement formulaire=Sur clic)
      Si(tabNoms#0)
  ` Utiliser le tableau tabIDs pour récupérer le bon enregistrement
         CHERCHER([Personnes];[Personnes]Numéro IDr=tabIDs{tabNoms})
  ` Traiter ici l'enregistrement
      Fin de si
Fin de cas

  ` Méthode objet du bouton bAversZ
  ` Tri croissant des tableaux en conservant la synchronisation
TRIER TABLEAU(tabNoms;tabIDs;>)

  ` Méthode objet du bouton bZversA
  ` Tri décroissant des tableaux en conservant la synchronisation
TRIER TABLEAU(tabNoms;tabIDs;<)
```

VALEURS DISTINCTES

VALEURS DISTINCTES (leChamp ; tableau {; tabNbVal})

Paramètre	Type		Description
leChamp	Champ	⇒	Champ à utiliser
tableau	Tableau	⇐	Tableau devant recevoir les données du champ indexable
tabNbVal	Tableau entier long, Tableau réel	⇐	Tableau devant recevoir le nombre d'occurrences de chaque valeur

Description

VALEURS DISTINCTES crée et remplit le tableau *tableau* avec toutes les valeurs distinctes provenant du champ *leChamp* pour la sélection courante de la table du champ et, optionnellement, retourne dans *tabNbVal* le nombre d'occurrences de chaque valeur.

Vous pouvez passer à cette commande tout type de champ **indexable**, c'est-à-dire dont le type supporte l'indexation mais qui n'est pas forcément indexé. Toutefois, l'exécution de la commande avec des champs non indexés est plus lente qu'avec des champs indexés. A noter également que dans ce cas, la commande perd l'enregistrement courant.
VALEURS DISTINCTES analyse et extrait les valeurs distinctes pour les enregistrements sélectionnés uniquement.

Note : Lorsque vous exécutez **VALEURS DISTINCTES** au sein d'une transaction non encore terminée, la commande tient compte des enregistrements créés au cours de la transaction.

Le tableau utilisé par **VALEURS DISTINCTES** doit être du même type que le champ passé en premier paramètre, sinon le tableau est retypé. Il y a une exception à cette règle : si le champ est de type Image (et est associé à un index de mots-clés), le tableau correspondant doit être de type Texte.

Après l'appel, la taille du tableau est égale au nombre de valeurs distinctes trouvées dans la sélection. La commande ne modifie pas la sélection courante ni l'enregistrement courant. Les éléments dans *tableau* sont triés par ordre croissant car **VALEURS DISTINCTES** utilise l'index du champ. Si cet ordre vous convient, vous n'avez donc pas besoin d'appeler **TRIER TABLEAU** après l'exécution de **VALEURS DISTINCTES**.

Note : Lorsque **VALEURS DISTINCTES** est exécutée avec un champ texte ou image associé à un index de mots-clés, la commande remplit le tableau avec les mots-clés de l'index. A la différence des autres types de données, les valeurs retournées diffèrent donc en fonction de l'existence de l'index. Dans le cas d'un champ texte, l'index de mots-clés est toujours pris en compte, même si le champ est également associé à un index standard. Si le champ texte ou image n'est pas associé à un index de mots-clés, le tableau est retourné vide.

La commande accepte en paramètre optionnel un tableau *tabNbVal*. Lorsqu'il est passé, ce tableau retourne, pour chaque valeur distincte de *leChamp* présente dans *tableau*, le nombre d'occurrences détectées dans la sélection courante. Le tableau *tabNbVal* est automatiquement dimensionné au même nombre d'éléments que *tableau*. Par exemple, pour une sélection qui contient trois enregistrements avec les valeurs de champs "A", "B" et "A", *tableau* contiendra {A;B} et *tabNbVal* contiendra {2;1}. Vous pouvez passer un tableau Entier long ou un tableau Réel dans *tabNbVal*.

Note : Le paramètre *tabNbVal* n'est pas pris en charge pour les champs texte ou image associés à des index de mots-clés (dans ce contexte, il est retourné vide).

ATTENTION : VALEURS DISTINCTES peut créer des tableaux de taille importante, en fonction de la taille de la sélection courante, ainsi que du type et de la taille des données à charger. Comme les tableaux résident en mémoire, il peut être utile de tester la taille des tableaux créés après l'exécution de la commande, ou d'utiliser une méthode projet d'interception d'erreurs installée par la commande **APPELER SUR ERREUR**.

4D Server : Cette commande est optimisée pour 4D Server. Le tableau est créé et les valeurs sont calculées sur le serveur. Seul le tableau est envoyé au client.

Note : Cette commande ne prend pas en charge les champs de type Objet.

Exemple 1

L'exemple suivant crée une liste de villes à partir de la sélection courante et indique à l'utilisateur le nombre de villes dans lesquelles la société dispose de magasins :

```
TOUT SELECTIONNER([Revendeurs]) ` Créer une sélection d'enregistrements
VALEURS DISTINCTES([Revendeurs]Ville;taVilles)
ALERTE("Cette société dispose de magasins dans "+Chaine(Taille tableau(taVilles))+" villes.")
```

Exemple 2

Vous souhaitez obtenir la liste complète des mots-clés contenus dans l'index des mots-clés du champ "Photos" :

```
TOUT SELECTIONNER([IMAGES])
TABLEAU TEXTE(<>_MesMotsCles;10)
VALEURS DISTINCTES([IMAGES]Photos;<>_MesMotsCles)
```

Exemple 3

Pour calculer des statistiques, vous voulez trier le nombre de valeurs distinctes d'un champ par ordre décroissant :

```
TABLEAU TEXTE($_issue_type;0)
TABLEAU ENTIER LONG($_issue_type_instance;0)
VALEURS DISTINCTES([Issue]iType;$_issue_type;$_issue_type_instances)
TRIER TABLEAU($_issue_type_instances;$_issue_type;<)
```

VALEURS DISTINCTES ATTRIBUT

VALEURS DISTINCTES ATTRIBUT (champObjet ; cheminAttribut ; tabValeurs)

Paramètre	Type		Description
champObjet	Champ	➡	Champ objet à utiliser
cheminAttribut	Texte	➡	Nom ou chemin de l'attribut dont vous voulez obtenir les valeurs distinctes
tabValeurs	Tableau texte, Tableau entier long, Tableau booléen, Tableau date, Tableau heure	⬅	Tableau des valeurs distinctes dans l'attribut

Description

La commande **VALEURS DISTINCTES ATTRIBUT** crée et remplit le tableau *tabValeurs* avec les valeurs uniques présentes dans l'attribut *cheminAttribut* du champ objet *champObjet* et ce, pour la sélection courante de la table à laquelle appartient le champ. Notez que le champ *champObjet* doit être type Objet, sinon une erreur est retournée. La commande peut être utilisée avec des champs indexés ou non indexés.

Passez un chemin valide d'attribut dans *cheminAttribut*. Utilisez la notation à points standard pour désigner le chemin d'attributs imbriqués, par exemple "société.adresse.numéro". Attention, gardez à l'esprit que les noms d'attributs d'objets tiennent compte de la casse des caractères.

Le tableau que vous passez dans *tabValeurs* doit être du même type que les valeurs stockées dans l'attribut *cheminAttribut*. Ces valeurs doivent être scalaires et peuvent être de type texte, numérique, booléen, date ou heure (les pointeurs, objets, BLOBs et images ne sont pas pris en charge). Assurez-vous que toutes les valeurs d'attributs du champ soient bien du même type, autrement une erreur est retournée. Par exemple, si l'attribut *cheminAttribut* contient "Lundi" dans un enregistrement et 10125 dans un autre enregistrement, **VALEURS DISTINCTES ATTRIBUT** retournera une erreur.

Si la commande est appelée pendant une transaction, les enregistrements créés dans la transation sont pris en compte.

Après l'exécution de la commande, la taille du tableau *tabValeurs* est égale au nombre de valeurs différentes trouvées dans la sélection.

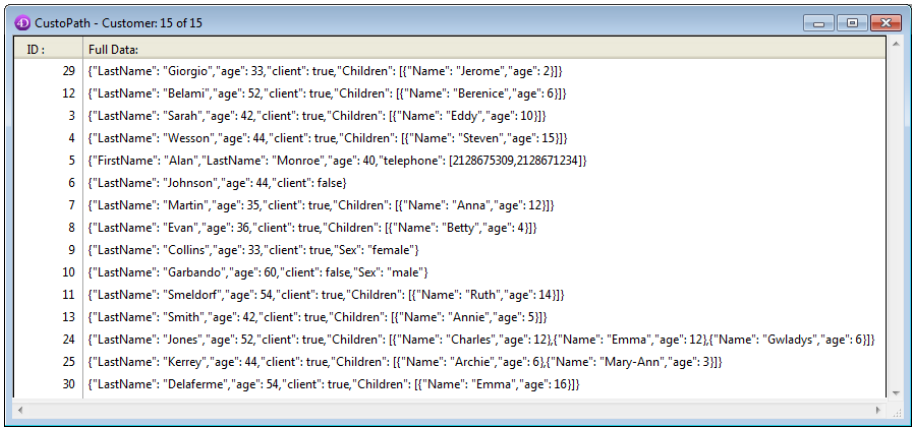
La commande ne modifie pas la sélection courante ni l'enregistrement courant.

Utilisation de la propriété virtuelle length

Vous pouvez utiliser la propriété virtuelle "length" avec cette commande. Cette propriété est automatiquement disponible pour tous les attributs de type tableau, et retourne la taille du tableau, c'est-à-dire le nombre d'éléments qu'il contient. Elle est destinée à une utilisation avec la commande **CHERCHER PAR ATTRIBUT** mais est également disponible pour **VALEURS DISTINCTES ATTRIBUT** afin d'obtenir les différentes tailles de tableaux pour un attribut.

Exemple

Votre base de données comporte un champ objet [Customer]full_Data avec 15 enregistrements :



Si vous exécutez ce code :

```
TABLEAU ENTIER LONG(aLAges;0)
TABLEAU ENTIER LONG(aLAgesChild;0)
TABLEAU ENTIER LONG(aLChildNum;0)
TOUT SELECTIONNER([Customer])
 //obtenir les valeurs distinctes de l'attribut "age"
VALEURS DISTINCTES ATTRIBUT([Customer]full_Data;"age";aLAges)
 //obtenir les valeurs distinctes de l'attribut dans le tableau Children"
VALEURS DISTINCTES ATTRIBUT([Customer]full_Data;"Children[].age";aLAgesChild)
 //obtenir les différents nombres d'enfants à l'aide de la propriété virtuelle length
VALEURS DISTINCTES ATTRIBUT([Customer]full_Data;"Children.length";aLChildNum)
```

Le tableau *aLAges* reçoit les éléments suivants :

Elément	Valeur
1	33
2	35
3	36
4	40
5	42
6	44
7	52
8	54
9	60

Le tableau *aLAgesChild* reçoit les éléments suivants :

Elément	Valeur
1	2
2	3
3	4
4	5
5	6
6	10
7	12
8	14
9	15
10	16

Le tableau *aLChildNum* reçoit les éléments suivants :

Elément	Valeur
1	1
2	2
3	3

VALEURS DISTINCTES CHEMINS

VALEURS DISTINCTES CHEMINS (champObjet ; tabChemins)

Paramètre	Type		Description
champObjet	Champ	➡	Champ objet indexé
tabChemins	Tableau texte	⬅	Tableau devant recevoir les chemins d'attributs du champ

Description

La commande **VALEURS DISTINCTES CHEMINS** retourne la liste des chemins d'attributs différents présents dans le champ objet indexé passé dans *champObjet* et ce, pour la sélection courante de la table à laquelle le champ appartient.

Vous devez passer dans *champObjet* un champ de type Objet indexé ; sinon, une erreur est retournée.

Après l'exécution de la commande, la taille du tableau *tabChemins* est égale au nombre de chemins différents trouvés dans la sélection. Les chemins des attributs imbriqués sont retournés avec la notation à points standard, par exemple "sociétés.adresse.numéro". Attention, gardez à l'esprit que les noms d'attributs d'objets tiennent compte de la casse des caractères.

Dans *tabChemins*, la liste des chemins d'attributs distincts est retournée dans l'ordre alphabétique (diacritique).

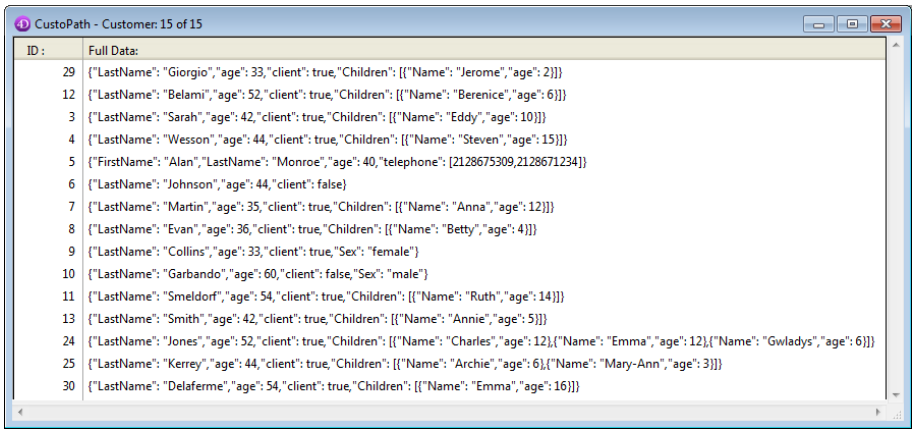
La commande ne modifie pas la sélection courante ni l'enregistrement courant.

Notes :

- Les enregistrements pour lesquels la valeur de *champObjet* est indéfinie ne sont pas pris en compte.
- Les chemins d'attributs créés pendant une transaction sont pris en compte par la commande. Attention, ces chemins sont conservés dans l'index du champ objet même si la transaction a été annulée.

Exemple

Votre base de données comporte un champ objet [Customer]full_Data avec 15 enregistrements. Le champ est indexé :



Si vous exécutez ce code :

```
TABLEAU TEXTE(aTPaths;0)
TOUT SELECTIONNER([Customer])
VALEURS DISTINCTES CHEMINS([Customer]full_Data;aTPaths)
```

Le tableau *aTPaths* reçoit les éléments suivants :

Elément	Valeur
1	"age"
2	"Children"
3	"Children[]"
4	"Children[].age"
5	"Children[].Name"
6	"Children.length"
7	"client"
8	"FirstName"
9	"LastName"
10	"Sex"
11	"telephone"
12	"telephone[]"
13	"telephone.length"

Note : "length" est une *propriété virtuelle* qui est disponible automatiquement pour tous les attributs de type tableau. Elle fournit la taille du tableau, c'est-à-dire le nombre d'éléments, et peut être utilisée dans les requêtes. Pour plus d'informations, reportez-vous au paragraphe **Utilisation de la propriété virtuelle length**.

_o_TABLEAU ALPHA

_o_TABLEAU ALPHA (longueurChaîne ; nomTableau ; taille {; taille2})

Paramètre	Type		Description
longueurChaîne	Entier long	⇒	Longueur de la chaîne (1..255)
nomTableau	Tableau	⇒	Nom du tableau
taille	Entier long	⇒	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Entier long	⇒	Nombre d'éléments des tableaux à deux dimensions

Note de compatibilité

Le fonctionnement de la commande **_o_TABLEAU ALPHA** est rigoureusement identique à la celui de commande **TABLEAU TEXTE** (le paramètre *longueurChaîne* est ignoré). Il est désormais conseillé d'utiliser exclusivement **TABLEAU TEXTE** dans vos développements 4D.

Texte multistyle

Le thème "**Texte multistyle**" regroupe les commandes du langage dédiées à la gestion et la modification des zones de texte multistyle, aussi appelées *zones de texte riche*.

Vous déclarez une zone de texte multistyle en activant l'option "Multistyle" dans la Liste des propriétés pour cette zone (cf. **LISTE SOURCES DONNEES** dans le manuel *Mode Développement*). La commande **OBJET Est un texte style** du thème "**Objets (Formulaires)**" permet de savoir si une zone de texte est en mode multistyle ou non.

Zones 4D Write Pro

La plupart des commandes du thème "**Texte multistyle**" prennent en charge les zones 4D Write Pro. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser les commandes du thème Texte multistyle** dans le guide de référence de 4D Write Pro.

- Notes de programmation
- Balises prises en charge
- ST CALCULER EXPRESSIONS
- ST FIGER EXPRESSIONS
- ST FIXER ATTRIBUTS
- ST FIXER OPTIONS
- ST FIXER TEXTE
- ST FIXER TEXTE BRUT
- ST INSERER EXPRESSION
- ST INSERER URL
- ST LIRE ATTRIBUTS
- ST Lire expression
- ST LIRE OPTIONS
- ST Lire texte
- ST Lire texte brut
- ST Lire type contenu
- ST LIRE URL

Commandes de gestion des objets texte

Les commandes permettant de manipuler des objets texte par programmation ne tiennent pas compte des éventuelles balises de style intégrées au texte. Elles agissent sur le texte affiché. Il s'agit des commandes suivantes :

- Thème **Interface utilisateur**
  SELECTIONNER TEXTE
  TEXTE SELECTIONNE

A noter que, lorsque vous utilisez ces commandes avec des commandes de manipulation des chaînes de caractères, il est nécessaire de filtrer les caractères de formatage à l'aide de la commande **ST Lire texte brut** :

```
SELECTIONNER TEXTE([Produits]Notes;1;Longueur(ST Lire texte brut([Produits]Notes))+1)
```

- Thème **Objets (Formulaires)**
  Les commandes permettant de modifier le style des objets (par exemple **OBJET FIXER POLICE**) s'appliquent à l'objet entier et non à la sélection. A noter que si l'objet n'a pas le focus au moment de l'exécution de la commande, la modification est appliquée simultanément à l'objet (la zone de texte) et à sa variable associée. Si l'objet a le focus, la modification est effectuée sur l'objet mais pas sur la variable associée. La modification n'est appliquée à la variable qu'au moment où l'objet perd le focus. Gardez ce principe à l'esprit lorsque vous programmez les zones de texte.

Si l'option "Stocker les balises par défaut" est cochée pour l'objet, l'utilisation de ces commandes provoquera une modification des balises enregistrées avec chaque objet.

Interaction des commandes génériques avec les textes multistyles

A compter de 4D v14, un nouveau mode d'interaction a été défini entre les commandes génériques telles que **OBJET FIXER COULEURS RVB** ou **OBJET FIXER STYLE POLICE** et les zones de texte multistyle.

Dans les versions précédentes de 4D, l'exécution d'une de ces commandes modifiait le contenu des balises de style personnalisées éventuellement insérées dans la zone. Désormais, seules les propriétés par défaut sont affectées par ces commandes (ainsi que les propriétés stockées via les balises par défaut, le cas échéant). Les balises de style personnalisées sont conservées telles quelles.

Par exemple, soit une zone multistyle dans laquelle les balises par défaut ont été stockées :

Ceci est un mot rouge

Le texte brut de la zone est le suivant :

```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#009900">Ceci est un mot <span
style="color:#D81E05">rouge</span></span>
```

Si vous exécutez le code suivant :

```
OBJET FIXER COULEUR(*;"maZone";-(Bleu+(256*Jaune)))
```

Avec 4D v14, la couleur rouge est préservée :

4D v14

Ceci est un mot rouge

```
<span style="text-align:left;font-
family:'Segoe UI';font-
size:9pt;color:#0000FF">Ceci est un mot
<span style="color:#D81E05">rouge</span>
</span>
```

versions précédentes

Ceci est un mot rouge

```
<span style="font-family:'Segoe UI';font-size:9pt;text-
align:left;font-weight:normal;font-style:normal;text-
decoration:none;color:#0000FF;"><span style="background-
color:#FFFFFF">Ceci est un mot rouge</span></span>
```

Les commandes génériques sont les suivantes :

OBJET FIXER COULEURS RVB
OBJET FIXER COULEUR
OBJET FIXER POLICE
OBJET FIXER STYLE POLICE
OBJET FIXER TAILLE POLICE

Dans le contexte des zones de texte multistyles, les commandes génériques doivent être utilisées pour définir les styles par défaut uniquement. Pour gérer les styles lors de l'exécution de la base, il est recommandé d'utiliser les commandes du thème "**Texte multistyle**".

Commande Lire texte edite

Lorsqu'elle est utilisée avec une zone de texte riche, la commande **Lire texte edite** (thème **Evénements formulaire**) retourne le texte de la zone courante en incluant les éventuelles balises de style.

Pour récupérer le texte "brut" (texte sans balises) en cours d'édition, vous devez utiliser la commande **ST Lire texte brut** :

```
ST Lire texte brut(Lire texte edite)
```

Commandes de recherche et de tri

Les recherches et les tris effectués parmi des objets multistyles tiennent compte des éventuelles balises de style enregistrées dans l'objet. Si une modification de style a été apportée à l'intérieur d'un mot, une recherche sur ce mot sera infructueuse.

Pour pouvoir effectuer des recherches et des tris valides, vous devez utiliser la commande **ST Lire texte brut**. Par exemple :

```
CHERCHER PAR FORMULE([MaTable];ST Lire texte brut([MaTable]MonchampStyle)="très bien")
```

Normalisation automatique des fins de lignes

Afin d'assurer la compatibilité multi-plate-forme des textes manipulés dans la base de données, 4D à compter de la v14 normalise automatiquement les fins de ligne afin qu'elles n'occupent qu'un seul caractère '\r' (retour chariot). Cette normalisation est effectuée au niveau des objets de formulaire hébergeant du texte multistyle ou du texte brut (variables ou champs). Les fins de ligne non natives ou utilisant un mélange de plusieurs caractères (par exemple '\r\n') sont considérées comme un seul '\r'.

A noter que, conformément à la norme XML (format des textes multistyles), les commandes de texte multistyle normalisent également les fins de ligne des variables texte non associées à des objets.

Ce principe facilite l'utilisation des commandes de texte multistyle ou du type **SELECTIONNER TEXTE** dans un contexte multi-plate-forme. Vous devez toutefois en tenir compte dans vos traitements si vous manipulez des textes de provenance hétérogène.

Balises prises en charge

Balises dynamiques

Les balises suivantes peuvent être utilisées dans les zones de texte multistyle de 4D.

Expression 4D

Cette balise permet d'insérer une expression 4D (expression, méthode, champ, variable, commande...) dans le texte. L'expression est stockée sous forme de référence (*token*) et est évaluée :

- à l'insertion de l'expression
- au chargement de l'objet
- à l'exécution de la commande **ST CALCULER EXPRESSIONS**
- à l'exécution de la commande **ST FIGER EXPRESSIONS** si le deuxième paramètre * est passé.

La valeur évaluée de l'expression n'est pas stockée dans la balise , seule sa référence l'est.

Note : Pour assurer une évaluation correcte de l'expression quelle que soit la langue ou la version de 4D, il est recommandé d'utiliser la syntaxe *tokenisée* pour les éléments dont le nom peut varier au fil des versions (commandes, tables, champs, constantes). Par exemple, pour insérer la commande **Heure courante**, saisissez '**Current time:C178**'. Pour plus d'informations sur ce point, reportez-vous à la section **Utiliser des tokens dans les formules**.

URL

Libellé visible

Cette balise permet d'insérer un URL dans le texte. Exemple :

```
<span><a href="http://www.4d.com/">Site Web de 4D</a></span>
```

Lien utilisateur

Cliquez ici

Les "liens utilisateurs" sont des liens ayant le même rendu visuel que les URLs, à la différence qu'un clic ne déclenche pas automatiquement l'ouverture de la source. Vous pouvez passer la chaîne que vous voulez comme référence. Il vous revient de programmer toute action personnalisée en réponse au clic. Ce principe vous permet de créer des liens qui ne sont pas des URLs mais des références de fichiers, de méthodes 4D, etc., que vous pouvez ouvrir ou exécuter en cas de clic. La commande **ST Lire type contenu** vous permet de détecter si un clic a eu lieu sur un lien utilisateur.

Les liens utilisateurs sont définis à l'aide de la commande **ST FIXER TEXTE**. Par exemple :

```
ST FIXER TEXTE(txtVar;"Ceci est un lien utilisateur: <span style=\"-d4-ref-user:'LienUtilisateur'\">Libellé du
lien utilisateur</span>";$début;$fin)
```

Balises personnalisées

Il est possible d'insérer toute balise dans le texte brut, par exemple <img src="http://doc.4d.com/images/ja.png">. Elles ne seront ni interprétées ni affichées mais conservées dans le code du texte brut. Cette possibilité est utile notamment dans le contexte d'emails formatés en HTML et incluant des images par exemple.

Balises de style

Ce paragraphe liste les attributs des balises <SPAN> pris en charge par 4D dans les zones de texte riche. Vous pouvez utiliser ces balises pour mettre en place une gestion personnalisée des styles. Seules les balises listées ci-dessous sont prises en charge par 4D pour les variations de style.

Nom de police

<SPAN STYLE="font-family: DESDEMONA"> ... </SPAN>

Taille de police

<SPAN STYLE="font-size: 20pt"> ... </SPAN>

Style de police

- **Gras**
  <SPAN STYLE="font-weight: bold"> ... </SPAN>
- **Italique ou normal**
  <SPAN STYLE="font-style: italic"> ... </SPAN>
  <SPAN STYLE="font-style: normal"> ... </SPAN>
- **Souligné**
  <SPAN STYLE="text-decoration: underline"> ... </SPAN>
- **Barré**
  <SPAN STYLE="text-decoration:line-through">...</SPAN>
  Note : Le style "barré" n'est pas pris en charge sous Mac OS. La balise correspondante peut toutefois être gérée par programmation.

Couleurs de police

<SPAN STYLE="color:green"> ... </SPAN>
ou
<SPAN STYLE="color:#006CCC">...</SPAN>

Couleurs de fond (Windows uniquement)

<SPAN STYLE="background-color:green"> ... </SPAN>
ou
<SPAN STYLE="background-color:#006CCC">...</SPAN>

Note : Sous Mac OS, cet attribut est ignoré. Il est supprimé en cas de modification de l'objet.

Valeurs de couleurs

Pour les attributs de couleur de police et couleur de fond, la valeur de couleur peut être soit le code hexadécimal des couleurs RVB, soit le nom d'une des 16 couleurs HTML définies pour le standard CSS par le W3C :

Color								
Name	Aqua	Black	Blue	Fuchsia	Gray	Green	Lime	Maroon
RGB	#00FFFF	#000000	#0000FF	#FF00FF	#808080	#008000	#00FF00	#800000

Color							
Name	Navy	Olive	Purple	Red	Silver	Teal	White	Yellow
RGB	#000080	#808000	#800080	#FF0000	#C0C0C0	#008080	#FFFFFF	#FFFF00

ST CALCULER EXPRESSIONS

ST CALCULER EXPRESSIONS ({* ;} objet {; débutSél {; finSél}})

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
débutSél	Entier long	➡	Début de la sélection
finSél	Entier long	➡	Fin de la sélection

Description

La commande **ST CALCULER EXPRESSIONS** met à jour les expressions 4D dynamiques situées dans le champ ou la variable de texte stylé désigné(e) par le paramètre *objet*.

Pour plus d'informations sur les expressions 4D utilisables dans les zone de texte multistyle, reportez-vous à la description de la commande **ST INSERER EXPRESSION**.

La commande réévalue le résultat des expressions présentes dans l'*objet* en fonction du contexte courant et affiche le résultat obtenu. Par exemple, si l'expression insérée est l'heure, la valeur sera modifiée à chaque appel de la commande **ST CALCULER EXPRESSIONS**. Les expressions sont également calculées :

- au moment de leur insertion
- au chargement de l'objet
- lorsqu'elles sont "figées" à l'aide de la commande **ST FIGER EXPRESSIONS**, si le deuxième paramètre * est passé.

ST CALCULER EXPRESSIONS ne modifie pas le texte stylé (contenant les balises *span*) mais uniquement le texte brut affiché dans *objet*. Les valeurs calculées ne sont pas stockées dans le texte stylé, seule leur référence y est stockée.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Il n'est pas nécessaire que *objet* ait le focus, en revanche, l'objet doit être inclus dans un formulaire, sinon la commande **ST CALCULER EXPRESSIONS** n'a pas d'effet.

Les paramètres optionnels *débutSél* et *finSél* vous permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style ou des références éventuellement présentes. A noter qu'une référence équivaut à un seul caractère.

- Si vous passez *débutSél* et *finSél*, **ST CALCULER EXPRESSIONS** met à jour uniquement les expressions situées à l'intérieur de cette sélection.
- Si vous passez uniquement *débutSél* ou si la valeur de *finSél* est supérieure au nombre total de caractères dans l'objet, toutes les expressions entre *débutSél* et la fin du texte sont calculées.
- Si vous omettez *débutSél* et *finSél*, toutes les expressions incluses dans la sélection utilisateur de *objet* sont calculées.

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Note : Si *débutSél* est supérieur à *finSél* (hormis si *finSél* vaut 0), la commande ne fait rien et la variable *OK* prend la valeur 0.

Exemple

Vous souhaitez mettre à jour les références incluses dans la sélection de texte :

```
ST CALCULER EXPRESSIONS(*;"monTexte";ST Début sélection;ST Fin sélection)
```

ST FIGER EXPRESSIONS

ST FIGER EXPRESSIONS ({* ;} objet {; débutSél {; finSél}}{; *})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
débutSél	Entier long	⇒	Début de la sélection
finSél	Entier long	⇒	Fin de la sélection
*	Opérateur	⇒	Si passé = mettre à jour les expressions avant de les figer

Description

La commande **ST FIGER EXPRESSIONS** "gèle" le contenu des expressions situées dans le champ ou la variable de texte stylé désigné(e) par le paramètre *objet*. Cette action convertit les expressions dynamiques en textes statiques et supprime de l'*objet* les références associées.

Pour plus d'informations sur les expressions 4D utilisables dans les zone de texte multistyle, reportez-vous à la description de la commande **ST INSERER EXPRESSION**.

La commande **ST FIGER EXPRESSIONS** vous permet de stocker la valeur calculée d'une expression à un instant donné. Cette opération est nécessaire notamment avant chaque utilisation de l'*objet* en-dehors d'une zone multistyle (exportation, stockage dans un fichier disque, impression...) car seule la référence de l'expression est conservée dans la zone.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Les paramètres optionnels *débutSél* et *finSél* vous permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style ou des références éventuellement présentes.

- Si vous passez *débutSél* et *finSél*, **ST FIGER EXPRESSIONS** fige uniquement les expressions situées à l'intérieur de cette sélection.
- Si vous passez uniquement *débutSél* ou si la valeur de *finSél* est supérieure au nombre total de caractères dans l'objet, toutes les expressions entre *débutSél* et la fin du texte sont figées.
- Si vous omettez *débutSél* et *finSél*, toutes les expressions incluses dans la sélection utilisateur de *objet* sont figées.

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Note : Si *débutSél* est supérieur à *finSél* (hormis si *finSél* vaut 0), la commande ne fait rien et la variable *OK* prend la valeur 0.

Par défaut, les expressions ne sont pas réévaluées avant d'être figées. Si vous souhaitez que les expressions soient recalculées puis figées, passez le second paramètre *.

Exemple

Vous souhaitez insérer l'heure courante au début du texte et la figer avant de stocker l'enregistrement :

```
    //Insertion de l'heure au début du texte
ST INSERER EXPRESSION(*;"StyledText_t";"Heure courante";1)
    //On fige l'expression
ST FIGER EXPRESSIONS(*;"StyledText_t";1)
```

ST FIXER ATTRIBUTS

ST FIXER ATTRIBUTS ({* ;} objet ; débutSél ; finSél ; nomAttribut ; valeurAttribut {; nomAttribut2 ; valeurAttribut2 ; ... ; nomAttributN ; valeurAttributN})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ Texte (si * est omis)
débutSél	Entier long	⇒	Début de la nouvelle sélection de texte
finSél	Entier long	⇒	Fin de la nouvelle sélection de texte
nomAttribut	Chaîne	⇒	Attribut à définir
valeurAttribut	Chaîne, Entier long	⇒	Nouvelle valeur d'attribut

Description

La commande **ST FIXER ATTRIBUTS** permet de modifier un ou plusieurs attribut(s) de style dans le ou les objet(s) de formulaire désigné(s) par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande s'applique uniquement à l'objet en cours d'édition et non à sa source de données (variable ou champ). Les modifications ne sont reportées dans la source (et donc dans les éventuels autres objets utilisant la même source) que lorsque l'objet en cours d'édition est validé via une perte de focus ou la touche **Entrée**. Si l'objet n'a pas le focus, la commande s'applique directement à la source de données et les modifications sont immédiatement répercutées aux éventuels autres objets utilisant la même source.
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. La commande s'applique directement au champ ou à la variable et les modifications sont répercutées à tous les objets utilisant cette source, y compris l'objet ayant le focus.

Note : Vous pouvez utiliser les attributs de style avec des champs de type Texte uniquement. Les champs de type Alpha ayant une longueur prédéfinie, l'ajout d'éventuelles balises de style entraînerait des pertes de données.

La définition d'un attribut s'effectue via l'insertion ou la modification de balises HTML de style dans à l'intérieur du texte (pour plus d'informations sur ce point, reportez-vous au manuel *Mode Développement*). A noter que **ST FIXER ATTRIBUTS** insère des balises de style dans tous les cas, même si *objet* désigne des objets texte de formulaire n'ayant pas la propriété Multistyle.

Les paramètres *débutSél* et *finSél* permettent de désigner la sélection de texte à laquelle appliquer la ou les modification(s) de style à l'intérieur de l'*objet*. Passez dans *débutSél* la position du premier caractère à modifier et dans *finSél* la position plus un du dernier caractère à modifier. Vous pouvez passer 0 dans *finSél* afin de désigner automatiquement le dernier caractère du texte (passez 1 dans *débutSél* pour désigner le premier caractère).
Si la valeur de *finSél* est supérieure au nombre de caractères de l'objet, tous les caractères entre *débutSél* et la fin du texte seront modifiés. Si *débutSél* est supérieur à *finSél* (hormis si *finSél* vaut 0), la commande ne fait rien et la variable OK prend la valeur 0.
Les valeurs *débutSél* et *finSél* ne tiennent pas compte des balises de style éventuellement déjà présentes dans la zone. Elles sont évaluées sur la base du texte brut (texte duquel les balises de style ont été filtrées).
4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Passez dans les paramètres *nomAttribut* et *valeurAttribut* respectivement le nom et la valeur de l'attribut à modifier. Vous pouvez passer autant de paires attribut/valeur que vous souhaitez. Pour définir le paramètre *nomAttribut*, utilisez les constantes prédéfinies placées dans le thème **Attributs de texte multistyle**. La valeur à passer dans le paramètre *valeurAttribut* dépend du paramètre *nomAttribut* :

Constante	Type	Valeur	Comment
Attribut couleur fond	Entier long	8	*valeurAttribut*=Valeur hexadécimale ou nom de couleur HTML (Windows uniquement)
Attribut couleur texte	Entier long	7	*valeurAttribut*=Valeur hexadécimale ou nom de couleur HTML
Attribut nom de police	Entier long	5	*valeurAttribut*=Nom de la famille de police (chaîne)
Attribut style barré	Entier long	3	*valeurAttribut*=0 : pas d'attribut barré pour la sélection *valeurAttribut*=1 : attribut barré pour la sélection
Attribut style gras	Entier long	1	*valeurAttribut*=0 : pas d'attribut gras pour la sélection *valeurAttribut*=1 : attribut gras pour la sélection
Attribut style italique	Entier long	2	*valeurAttribut*=0 : pas d'attribut italique pour la sélection *valeurAttribut*=1 : attribut italique pour la sélection
Attribut style souligné	Entier long	4	*valeurAttribut*=0 : pas d'attribut souligné pour la sélection *valeurAttribut*=1 : attribut souligné pour la sélection
Attribut taille texte	Entier long	6	*valeurAttribut*=Nombre de points (numérique)

Couleurs
Si vous passez la constante Attribut couleur texte ou Attribut couleur fond dans *nomAttribut*, vous devez passer dans *valeurAttribut* une chaîne contenant soit un nom de couleur HTML soit une valeur de couleur hexadécimale :

Nom de couleur HTML	Valeur hexadécimale
Aqua	#00FFFF
Black	#000000
Blue	#0000FF
Fushia	#FF00FF
Gray	#808080
Green	#008000
Lime	#00FF00
Maroon	#800000
Navy	#000080
Olive	#808000
Purple	#800080
Red	#FF0000
Silver	#C0C0C0
Teal	#008080
White	#FFFFFF
Yellow	#FFFF00

Exemple

Dans cet exemple, nous modifions la taille, la couleur de texte ainsi que les attributs gras et souligné des caractères 2 à 4 du champ :

```
ST FIXER ATTRIBUTS([MaTable]MonChamp;2;5;Attribut nom de police;"Arial";Attribut taille texte;10;Attribut style souligné;1;Attribut style gras;1;Attribut couleur texte;"Blue")
```

Variables et ensembles système

A l'issue de l'exécution de cette commande, la variable OK prend la valeur 1 si aucune erreur n'a été rencontrée et 0 dans le cas contraire. C'est le cas notamment lorsque l'évaluation des balises de style échoue (balise incorrecte ou manquante).

A noter qu'en cas d'erreur, la variable n'est pas modifiée. Lorsqu'une erreur se produit sur une variable lors de l'évaluation du texte, 4D transforme le texte en texte brut ; par conséquent, les caractères <, > et & seront convertis en entités HTML.

ST FIXER OPTIONS

ST FIXER OPTIONS ({* ;} objet ; option ; valeur {; option2 ; valeur2 ; ... ; optionN ; valeurN})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
option	Entier long	⇒	Option à définir
valeur	Entier long	⇒	Nouvelle valeur de l'option

Description

La commande **ST FIXER OPTIONS** vous permet de modifier une ou plusieurs options de fonctionnement du champ ou de la variable de texte stylé désigné(e) par le paramètre *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans *option* le code de l'option à modifier et dans *valeur*, sa nouvelle valeur.

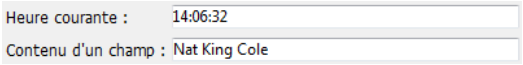
Le paramètre *option* prend en charge la constante suivante du thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Mode affichage expressions	Entier long	1	Le paramètre *valeur* peut contenir ST Valeurs or ST Références

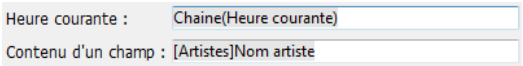
Vous pouvez passer dans le paramètre *valeur* l'une des constantes suivantes :

Constante	Type	Valeur	Comment
ST Références	Entier long	1	Affichage des chaînes source des expressions
ST Valeurs	Entier long	0	Affichage des valeurs calculées des expressions

Affichage des valeurs :



Affichage des expressions :



Exemple

Le code suivant vous permet de basculer le mode d'affichage de la zone :

```
ST LIRE OPTIONS(*;"StyledText_t";ST Mode affichage expressions;$valueExpr)
Si($valueExpr=1)
      ST FIXER OPTIONS(*;"StyledText_t";ST Mode affichage expressions;ST Valeurs)
Sinon
      ST FIXER OPTIONS(*;"StyledText_t";ST Mode affichage expressions;ST Références)
Fin de si
```

ST FIXER TEXTE

ST FIXER TEXTE ({* ;} objet ; nouvTexte {; débutSél {; finSél}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ Texte (si * est omis)
nouvTexte	Texte	⇒	Texte multistyle à insérer
débutSél	Entier long	⇒	Début de la sélection
finSél	Entier long	⇒	Fin de la sélection

Description

La commande **ST FIXER TEXTE** insère le texte passé dans le paramètre *nouvTexte* dans le champ ou la variable de texte multistyle désigné(e) par le paramètre *objet*. Cette commande s'applique uniquement au texte brut du paramètre *objet*, sans modifier les éventuelles balises de style qu'il contient. Elle permet de modifier par programmation du texte multistyle affiché à l'écran.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande s'applique uniquement à l'objet en cours d'édition et non à sa source de données (variable ou champ). Les modifications ne sont reportées dans la source (et donc dans les éventuels autres objets utilisant la même source) que lorsque l'objet en cours d'édition est validé via une perte de focus ou la touche **Entrée**. Si l'objet n'a pas le focus, la commande s'applique directement à la source de données et les modifications sont immédiatement répercutées aux éventuels autres objets utilisant la même source.
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. La commande s'applique directement au champ ou à la variable et les modifications sont répercutées à tous les objets utilisant cette source, y compris l'objet ayant le focus.

Passez dans *nouvTexte* le texte à insérer. La commande **ST FIXER TEXTE** est destinée aux manipulations de texte stylé (multistyle), contenant des balises de type . Dans tous les autres cas (notamment en cas de manipulation de texte non stylé mais contenant les caractères <, > ou &), vous devez utiliser la commande **ST FIXER TEXTE BRUT**. Si vous passez à la commande **ST FIXER TEXTE** un texte brut contenant des caractères <, > ou &, la commande ne fait rien. Ce principe de fonctionnement est nécessaire car l'insertion directe d'une chaîne telle que "a<b" au sein d'un texte stylé va fausser l'analyse interne des balises . Dans ce cas, le caractère "<" doit être préalablement encodé "<", ce qui est effectué par la commande **ST FIXER TEXTE BRUT** (voir également l'exemple de cette commande).

Les paramètres optionnels *débutSel* et *finSel* vous permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style éventuellement présentes dans le texte. L'action de la commande diffère en fonction des paramètres facultatifs *débutSél* et *finSél* :

- si vous omettez *débutSél* et *finSél*, **ST FIXER TEXTE** remplace la totalité du texte de *objet* par *nouvTexte*,
- si vous passez uniquement *débutSél* ou si les valeurs de *débutSél* et *finSél* sont égales, **ST FIXER TEXTE** insère le texte *nouvTexte* dans *objet* à partir de *débutSél*.
- si vous passez *débutSél* et *finSél*, **ST FIXER TEXTE** remplace le texte brut défini par ces bornes avec le texte *nouvTexte*.
- vous pouvez passer 0 dans *finSél* afin de désigner automatiquement le dernier caractère du texte (passez 1 dans *débutSél* pour désigner le premier caractère).

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Note : Si *débutSél* est supérieur à *finSél*, le texte n'est pas modifié et la variable OK prend la valeur 0 (hormis lorsque *finSél* vaut 0, cf. ci-dessus).

Variables et ensembles système

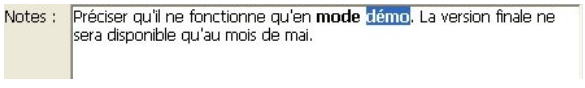
A l'issue de l'exécution de cette commande, la variable OK prend la valeur 1 si aucune erreur n'a été rencontrée et 0 dans le cas contraire. C'est le cas notamment lorsque l'évaluation des balises de style échoue (balise incorrecte ou manquante).

A noter qu'en cas d'erreur, la variable n'est pas modifiée. Lorsqu'une erreur se produit sur une variable lors de l'évaluation du texte, 4D transforme le texte en texte brut ; par conséquent, les caractères <, > et & seront convertis en entités HTML.

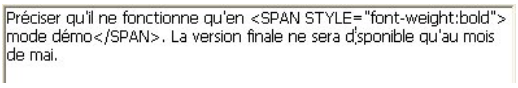
Exemple 1

Vous souhaitez remplacer le texte multistyle sélectionné par l'utilisateur avec le contenu d'une variable.

Voici le texte sélectionné :



Le contenu stocké dans le champ est le suivant :



Après exécution de ce code :

```
vtempo:="Démonstration"
TEXTE SELECTIONNE([Produits]Notes;vDebut;vFin)
ST FIXER TEXTE([Produits]Notes;vtempo;vDebut;vFin)
```

Le champ et son contenu sont les suivants :

Exemple 2

Reportez-vous à l'exemple de la commande **ST FIXER TEXTE BRUT**.

ST FIXER TEXTE BRUT

ST FIXER TEXTE BRUT ({* ;} objet ; nouvTexte {; débutSél {; finSél}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
nouvTexte	Texte	⇒	Texte brut à insérer
débutSél	Entier long	⇒	Début de la sélection
finSél	Entier long	⇒	Fin de la sélection

Description

La commande **ST FIXER TEXTE BRUT** insère le texte passé dans le paramètre *nouvTexte* dans le champ ou la variable de texte multistyle désigné(e) par le paramètre *objet*. Cette commande s'applique uniquement au texte brut du paramètre *objet*, sans modifier les éventuelles balises de style qu'il contient.

A la différence de la commande **ST FIXER TEXTE**, **ST FIXER TEXTE BRUT** permet d'insérer uniquement du texte sans style. Le texte *nouvTexte* ne doit pas contenir de balises de style. S'il contient les caractères <, > ou &, ils seront considérés comme des caractères standard et seront convertis en entités HTML :

- '&' est converti en &
- '<' est converti en <
- '>' est converti en >

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande s'applique uniquement à l'objet en cours d'édition et non à sa source de données (variable ou champ). Les modifications ne sont reportées dans la source (et donc dans les éventuels autres objets utilisant la même source) que lorsque l'objet en cours d'édition est validé via une perte de focus ou la touche **Entrée**. Si l'objet n'a pas le focus, la commande s'applique directement à la source de données et les modifications sont immédiatement répercutées aux éventuels autres objets utilisant la même source.
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. La commande s'applique directement au champ ou à la variable et les modifications sont répercutées à tous les objets utilisant cette source, y compris l'objet ayant le focus.

Passez dans *nouvTexte* le texte brut à insérer.

Les paramètres optionnels *débutSél* et *finSél* vous permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style éventuellement présentes dans le texte. L'action de la commande diffère en fonction des paramètres facultatifs *débutSél* et *finSél* :

- si vous omettez *débutSél* et *finSél*, **ST FIXER TEXTE BRUT** remplace la totalité du texte de *objet* par *nouvTexte*,
- si vous passez uniquement *débutSél* ou si les valeurs de *débutSél* et *finSél* sont égales, **ST FIXER TEXTE BRUT** insère le texte *nouvTexte* dans *objet* à partir de *débutSél*,
- si vous passez *débutSél* et *finSél*, **ST FIXER TEXTE BRUT** remplace le texte brut défini par ces bornes avec le texte *nouvTexte*.
- vous pouvez passer 0 dans *finSél* afin de désigner automatiquement le dernier caractère du texte (passez 1 dans *débutSél* pour désigner le premier caractère).

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

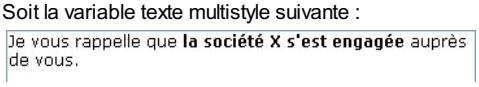
(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Le style du premier caractère remplacé sera utilisé pour la totalité du texte *nouvTexte*.

Si *débutSél* est supérieur à *finSél*, le texte n'est pas modifié et la variable OK prend la valeur 0 (hormis lorsque *finSél* vaut 0, cf. ci-dessus).

Exemple

Soit la variable texte multistyle suivante :



Vous voulez insérer des noms de sociétés stockés dans un champ texte. Ces noms peuvent comporter par exemple le caractère "&". Dans ce cas, il est nécessaire d'utiliser la commande **ST FIXER TEXTE BRUT** :

```
ST FIXER TEXTE BRUT(monTexteStyl;[Société]Nom;33;34)
```

Le résultat est alors :

Voici le texte brut contenu dans la variable :

```
Je vous rappelle que <SPAN STYLE="font-weight:bold">
la société </SPAN><SPAN STYLE="font-weight:bold">
Smith &amp;amp; Jones</SPAN><SPAN STYLE="font-
weight:bold"> s'est engagée</SPAN> auprès de vous.
```

Vous pouvez constater que le texte inséré a été encapsulé au sein d'une paire de balises de style supplémentaires. Ces balises correspondent au style du caractère précédent l'insertion. Ce mécanisme permet de garantir un affichage correct des champs multistyles dans tous les cas de figure.

Note : Si vous aviez utilisé la commande **ST FIXER TEXTE** dans ce cas, 4D n'aurait rien inséré, car la présence du caractère "&" non encodé empêcherait l'interprétation des balises de style présentes dans la variable. Pour plus d'informations, reportez-vous à la description de cette commande.

Variables et ensembles système

A l'issue de l'exécution de cette commande, la variable OK prend la valeur 1 si aucune erreur n'a été rencontrée et 0 dans le cas contraire. C'est le cas notamment lorsque l'évaluation des balises de style échoue (balise incorrecte ou manquante).

A noter qu'en cas d'erreur, la variable n'est pas modifiée. Lorsqu'une erreur se produit sur une variable lors de l'évaluation du texte, 4D transforme le texte en texte brut ; par conséquent, les caractères <, > et & seront convertis en entités HTML.

ST INSERER EXPRESSION

ST INSERER EXPRESSION ({* ;} objet ; expression {; débutSél {; finSél}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
expression	Texte	⇒	Expression et (optionnel) format à insérer
débutSél	Entier long	⇒	Début de la sélection
finSél	Entier long	⇒	Fin de la sélection

Description

La commande **ST INSERER EXPRESSION** insère une référence à l'*expression* dans le champ ou la variable de texte multistyle désigné(e) par le paramètre *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans le paramètre *expression* l'expression 4D à évaluer dans l'*objet*. Une expression 4D valide est une chaîne retournant une valeur. *expression* peut être un champ, une variable, une commande 4D, une instruction retournant une valeur, une méthode projet, etc.

L'expression doit être passée entre guillemets ("").

Note : Le paramètre *expression* ne peut pas être une variable de type Image.

Si *expression* retourne une valeur contenant des retours chariot et des tabulations, 4D formate le texte en fonction de l'objet hébergeant l'expression ; les caractères retours chariot sont interprétés comme des retours à la ligne.

Vous pouvez formater l'expression en incluant une information de formatage dans le paramètre *expression*. Dans ce cas, le paramètre doit être de la forme :

```
"Chaine(valeur;format)"
```

... où *valeur* contient l'expression elle-même et *format* le formatage à appliquer. Le paramètre *format* peut contenir les valeurs suivantes :

- pour les numériques : tout format d'affichage numérique existant ou non, par exemple "###,##"
- pour les dates : un nombre désignant un format de date existant. Vous pouvez utiliser les constantes du thème "**Formats d'affichage des dates**", par exemple Système date court.
- pour les heures : un nombre désignant un format d'heure existant. Vous pouvez utiliser les constantes du thème "**Formats d'affichage des heures**", par exemple Système heure court.

Par exemple :

```
"Chaine([Table_1]Champ_1;Système date court)"
```

Par défaut, les **valeurs** des expressions sont affichées dans les zones de texte multistyle. Vous pouvez forcer l'affichage des **références** à l'aide de la commande **ST FIXER OPTIONS**.

Les paramètres optionnels *débutSél* et *finSél* permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style éventuellement présentes dans le texte.

- Si vous passez uniquement *débutSél*, le résultat de l'expression est inséré à l'emplacement spécifié.
- Si vous omettez *débutSél* et *finSél*, le résultat de l'expression est inséré à l'emplacement du curseur.
- Si vous passez *débutSél* et *finSél*, **ST INSERER EXPRESSION** remplace le texte situé à l'intérieur de cette sélection par le résultat de l'*expression*. Si la valeur de *finSél* est supérieure au nombre total de caractères dans l'objet, tous les caractères entre *débutSél* et la fin du texte sont remplacés par le résultat de l'*expression*.

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Note : Si *débutSél* est supérieur à *finSél* (hormis si *finSél* vaut 0), la commande ne fait rien et la variable *OK* prend la valeur 0.

Exemple

Vous souhaitez remplacer le texte sélectionné par le résultat d'une méthode projet :

```
ST INSERER EXPRESSION(*;"myText";"MyMethod";ST Début sélection;ST Fin sélection)
```

ST INSERER URL

ST INSERER URL ({* ;} objet ; texteURL ; adresseURL {; débutSél {; finSél}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
texteURL	Texte	⇒	Libellé visible de l'URL
adresseURL	Texte	⇒	Adresse de l'URL
débutSél	Entier long	⇒	Début de la sélection
finSél	Entier long	⇒	Fin de la sélection

Description

La commande **ST INSERER URL** insère un lien URL dans le champ ou la variable de texte stylé désigné(e) par le paramètre *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

Passez dans le paramètre *texteURL* le libellé visible de l'URL, tel qu'il doit apparaître dans l'objet. Par exemple, des libellés comme "Site Web de 4D" ou "Suivez ce lien pour plus d'informations" peuvent être utilisés. Vous pouvez également utiliser l'adresse elle-même, par exemple "http://www.4d.com".

Passez dans le paramètre *adresseURL* l'adresse complète à laquelle connecter la page du navigateur, par exemple "http://www.4D.com".

Les paramètres optionnels *débutSél* et *finSél* permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style éventuellement présentes dans le texte.

- Si vous passez uniquement *débutSél*, *texteURL* est inséré à l'emplacement spécifié.
- Si vous omettez *débutSél* et *finSél*, *texteURL* est inséré à l'emplacement du curseur.
- Si vous passez *débutSél* et *finSél*, **ST INSERER URL** remplace le texte situé à l'intérieur de cette sélection par *texteURL*. Si la valeur de *finSél* est supérieure au nombre total de caractères dans l'objet, tous les caractères entre *débutSél* et la fin du texte sont remplacés par *texteURL*.

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Note : Si *débutSél* est supérieur à *finSél* (hormis si *finSél* vaut 0), la commande ne fait rien et la variable *OK* prend la valeur 0.

Une fois le lien inséré, il est actif : l'action **Ctrl+clic** (Windows) ou **Commande+clic** (OS X) sur le lien ouvre une page du navigateur par défaut à l'adresse définie dans le paramètre *adresseURL*.

Exemple

Vous souhaitez insérer un lien vers le site Web de 4D à la place de la sélection de texte dans l'objet "myText" :

```
vTitle:="4D Web Site"
vURL:="http://www.4d.com/"
ST INSERER URL(*;"myText";vTitle;vURL;ST Début sélection;ST Fin sélection)
```

ST LIRE ATTRIBUTS

ST LIRE ATTRIBUTS ({* ;} objet ; débutSél ; finSél ; nomAttribut ; valeurAttribut {; nomAttribut2 ; valeurAttribut2 ; ... ; nomAttributN ; valeurAttributN})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
débutSél	Entier long	⇒	Début de la sélection de texte
finSél	Entier long	⇒	Fin de la sélection de texte
nomAttribut	Entier long	⇒	Attribut à lire
valeurAttribut	Variable	⇐	Valeur courante de l'attribut

Description

La commande **ST LIRE ATTRIBUTS** permet de récupérer la valeur courante d'un attribut de style dans une sélection de texte du ou des objet(s) de formulaire désigné(s) par *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande retourne les informations de l'objet en cours d'édition ; si l'objet n'a pas le focus, la commande retourne les informations de la source de données de l'objet (variable ou champ).
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. A l'exécution, la commande retourne les informations de la variable ou du champ.

Les paramètres *débutSél* et *finSél* permettent de désigner la sélection de texte de laquelle lire l'attribut de style. Passez dans *débutSél* la position du premier caractère et dans *finSél* la position plus un du dernier caractère de la sélection. Vous pouvez passer 0 dans *finSél* afin de désigner automatiquement le dernier caractère du texte (passez 1 dans *débutSél* pour désigner le premier caractère).
Si les valeurs de *débutSél* et *finSél* sont égales ou si *débutSél* est supérieur à *finSél* (hormis si *finSél* vaut 0), une erreur est retournée.
Les valeurs *débutSél* et *finSél* ne tiennent pas compte des balises de style éventuellement déjà présentes dans la zone. Elles sont évaluées sur la base du texte brut (texte duquel les balises de style ont été filtrées). 4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Passez dans le paramètre *nomAttribut* le nom de l'attribut à lire et dans le paramètre *valeurAttribut* une variable devant récupérer la valeur courante de l'attribut. Pour définir le paramètre *nomAttribut*, vous devez utiliser l'une des constantes du thème **Attributs de texte multistyle**.

Constante	Type	Valeur	Comment
Attribut couleur fond	Entier long	8	*valeurAttribut*=Valeur hexadécimale ou nom de couleur HTML (Windows uniquement)
Attribut couleur texte	Entier long	7	*valeurAttribut*=Valeur hexadécimale ou nom de couleur HTML
Attribut nom de police	Entier long	5	*valeurAttribut*=Nom de la famille de police (chaîne)
Attribut style barré	Entier long	3	*valeurAttribut*=0 : pas d'attribut barré pour la sélection *valeurAttribut*=1 : attribut barré pour la sélection
Attribut style gras	Entier long	1	*valeurAttribut*=0 : pas d'attribut gras pour la sélection *valeurAttribut*=1 : attribut gras pour la sélection
Attribut style italique	Entier long	2	*valeurAttribut*=0 : pas d'attribut italique pour la sélection *valeurAttribut*=1 : attribut italique pour la sélection
Attribut style souligné	Entier long	4	*valeurAttribut*=0 : pas d'attribut souligné pour la sélection *valeurAttribut*=1 : attribut souligné pour la sélection
Attribut taille texte	Entier long	6	*valeurAttribut*=Nombre de points (numérique)

Vous pouvez passer autant de paires attribut/valeur que vous souhaitez.

Si la valeur de l'attribut *nomAttribut* est identique dans la totalité de la sélection, elle est retournée dans *valeurAttribut*. Si cette valeur est différente ou si *objet* ne contient pas de balises SPAN, les valeurs suivantes sont retournées :

nomAttribut	valeurAttribut si attribut hétérogène dans la sélection ou pas de balises SPAN
Attribut couleur fond	FFFFFFFF
Attribut couleur texte	FFFFFFFF
Attribut nom de police	"" (chaîne vide)
Attribut style barré	2
Attribut style gras	2
Attribut style italique	2
Attribut style souligné	2
Attribut taille texte	-1

Exemple

Soit un champ [Table_1]StyledText affiché dans un formulaire. L'objet comporte la propriété Multistyle et est nommé "StyledText_t". Vous souhaitez

récupérer le texte sélectionné ainsi que le statut de l'attribut Gras. Vous pouvez procéder de deux manières différentes, selon que vous utilisez le nom d'objet ou la référence de champ.

- Utilisation du nom d'objet :

```
$text:=ST Lire texte(*;"StyledText_t";ST Début sélection;ST Fin sélection)
ST LIRE ATTRIBUTS(*;"StyledText_t";ST Début sélection;ST Fin sélection;Attribut style gras;$bold)
```

- Utilisation du nom de champ :

```
TEXTE SELECTIONNE([Table_1]StyledText;$Begin_l;$End_l)
$text:=ST Lire texte([Table_1]StyledText;$Begin_l;$End_l)
ST LIRE ATTRIBUTS([Table_1]StyledText;$Begin_l;$End_l;Attribut style gras;$bold)
```

Variables et ensembles système

A l'issue de l'exécution de cette commande, la variable OK prend la valeur 1 si aucune erreur n'a été rencontrée et 0 dans le cas contraire. C'est le cas notamment lorsque l'évaluation des balises de style échoue (balise incorrecte ou manquante).

A noter qu'en cas d'erreur, la variable n'est pas modifiée. Lorsqu'une erreur se produit sur une variable lors de l'évaluation du texte, 4D transforme le texte en texte brut ; par conséquent, les caractères <, > et & seront convertis en entités HTML.

ST Lire expression

ST Lire expression ({* ;} objet {; débutSél {; finSél}}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
débutSél	Entier long	⇒	Début de la sélection
finSél	Entier long	⇒	Fin de la sélection
Résultat	Texte	⇐	Libellé de l'expression

Description

La commande **ST Lire expression** retourne la première expression présente dans la sélection courante du champ ou de la variable de texte multistyle désigné(e) par le paramètre *objet*.

La commande retourne le libellé de l'expression tel qu'il a été inséré dans l'objet (par exemple "maméthode" ou "[table1]champ1"). La valeur courante de l'expression n'est pas retournée.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande retourne les informations de l'objet en cours d'édition ; si l'objet n'a pas le focus, la commande retourne les informations de la source de données de l'objet (variable ou champ).
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. A l'exécution, la commande retourne les informations de la variable ou du champ.

Les paramètres optionnels *débutSél* et *finSél* permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style éventuellement présentes dans le texte.

- Si vous passez *débutSél* et *finSél*, **ST Lire expression** recherche l'expression à l'intérieur de cette sélection.
- Si vous passez uniquement *débutSél* ou si la valeur de *finSél* est supérieure au nombre total de caractères dans l'*objet*, la commande recherche l'expression entre *débutSél* et la fin du texte.
- Si vous omettez *débutSél* et *finSél*, la commande recherche l'expression à l'intérieur de la sélection courante de texte.

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Note : Si *débutSél* est supérieur à *finSél* (hormis si *finSél* vaut 0), la commande ne fait rien et la variable *OK* prend la valeur 0.

Si aucune expression n'est présente dans la sélection, la commande retourne une chaîne vide.

Exemple 1

Sur un événement double-clic, vous vérifiez que vous êtes bien en présence d'une expression, et dans ce cas affichez un dialogue où vous avez récupéré ses valeurs afin de permettre à l'utilisateur de la modifier :

```
Au cas ou
   :(Evenement formulaire=Sur double clic)
      TEXTE SELECTIONNE(*;"StyledText_t";startSel;endSel)
      Si(ST Lire type contenu(*;"StyledText_t";startSel;endSel)=ST Type expression)
         vExpression:=ST Lire expression(*;"StyledText_t";startSel;endSel)
         $winRef:=Creer fenetre formulaire("Dial_InsertExpr";Form dialogue modal déplaçable;Centrée
horizontalement;Centrée verticalement;*)
         DIALOGUE("Dial_InsertExpr")
         Si(OK=1)
            ST INSERER EXPRESSION(*;"StyledText_t";vExpression;startSel;endSel)
            SELECTIONNER TEXTE(*;"StyledText_t";startSel;endSel)
         Fin de si
      Fin de si
Fin de cas
```

Exemple 2

Vous souhaitez exécuter une méthode 4D en réponse à un clic sur un lien utilisateur :

```
Au cas ou
   :(Evenement formulaire=Sur clic)
            //on récupère la sélection
      TEXTE SELECTIONNE(*;"myText";startSel;endSel)
      Si(startSel#endSel)  //il y a du contenu sélectionné
```

```
                //on lit le type de contenu
            $CT_type:=ST Lire type contenu(*;"myText";startSel;endSel)
            Si($CT_type=ST Type utilisateur) //c'est un lien utilisateur
               MaMethode //on exécute une méthode 4D
            Fin de si
         Fin de si
   Fin de cas
```

ST LIRE OPTIONS

ST LIRE OPTIONS ({* ;} objet ; option ; valeur {; option2 ; valeur2 ; ... ; optionN ; valeurN})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
option	Entier long	⇒	Option à lire
valeur	Entier long	⇐	Valeur courante de l'option

Description

La commande **ST LIRE OPTIONS** permet d'obtenir la valeur courante d'une ou plusieurs options de fonctionnement du champ ou de la variable de texte multistyle désigné(e) par le paramètre *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande retourne les informations de l'objet en cours d'édition ; si l'objet n'a pas le focus, la commande retourne les informations de la source de données de l'objet (variable ou champ).
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. A l'exécution, la commande retourne les informations de la variable ou du champ.

Passez dans *option* le code de l'option à lire. La commande retourne dans *valeur* la valeur courante de l'option. Pour ces deux paramètres, vous pouvez utiliser les constantes suivantes du thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Mode affichage expressions	Entier long	1	Le paramètre *valeur* peut contenir ST Valeurs or ST Références
ST Références	Entier long	1	Affichage des chaînes source des expressions
ST Valeurs	Entier long	0	Affichage des valeurs calculées des expressions

ST Lire texte

ST Lire texte ({* ;} objet {; débutSél {; finSél}}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ Texte (si * est omis)
débutSél	Entier long	⇒	Début de la sélection
finSél	Entier long	⇒	Fin de la sélection
Résultat	Texte	⇐	Texte incluant les balises de style

Description

La commande **ST Lire texte** retourne le texte multistyle présent dans le champ ou la variable de texte désigné(e) par le paramètre *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande retourne les informations de l'objet en cours d'édition ; si l'objet n'a pas le focus, la commande retourne les informations de la source de données de l'objet (variable ou champ).
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. A l'exécution, la commande retourne les informations de la variable ou du champ.

La commande retourne le texte avec les éventuelles balises de style qui lui sont associées, ce qui permet par exemple de copier et coller du texte en conservant les styles.

Les paramètres optionnels *débutSél* et *finSél* vous permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style éventuellement présentes.

- si vous omettez *débutSél* et *finSél*, **ST Lire texte** retourne la totalité du texte contenu dans *objet*,
- si vous passez *débutSél* et *finSél*, **ST Lire texte** retourne la sélection de texte définie par ces bornes.

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Si les valeurs de *débutSél* et *finSél* sont égales ou si *débutSél* est supérieur à *finSél*, une erreur est retournée.

Variables et ensembles système

A l'issue de l'exécution de cette commande, la variable OK prend la valeur 1 si aucune erreur n'a été rencontrée et 0 dans le cas contraire. C'est le cas notamment lorsque l'évaluation des balises de style échoue (balise incorrecte ou manquante).

A noter qu'en cas d'erreur, la variable n'est pas modifiée. Lorsqu'une erreur se produit sur une variable lors de l'évaluation du texte, 4D transforme le texte en texte brut ; par conséquent, les caractères <, > et & seront convertis en entités HTML.

ST Lire texte brut

ST Lire texte brut ({* ;} objet {; modeRéf}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable ou un champ
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable ou champ (si * est omis)
modeRéf	Entier long	⇒	Mode de prise en charge des références présentes dans le texte
Résultat	Texte	⇐	Texte sans balises

Description

La commande **ST Lire texte brut** supprime toute balise de style du champ ou de la variable texte désigné(e) par les paramètres * et *objet*, et retourne le texte brut.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande retourne les informations de l'objet en cours d'édition ; si l'objet n'a pas le focus, la commande retourne les informations de la source de données de l'objet (variable ou champ).
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. A l'exécution, la commande retourne les informations de la variable ou du champ.

Le paramètre optionnel *modeRéf* permet d'indiquer la manière dont les références présentes dans *objet* doivent être retournées. Passez dans *modeRéf* une des constantes suivantes, placées dans le thème "**Texte multistyle**" (vous pouvez passer une constante ou une combinaison de constantes) :

Constante	Type	Valeur	Comment
ST Balises comme code xml	Entier long	128	Le code XML de la balise est retourné en texte brut. Par exemple pour la balise '<img src="test.jpg" alt="image">mon image</img>', le texte brut est '<img src="test.jpg" alt="image">mon image</img>'
ST Balises comme texte brut	Entier long	64	Le libellé de la balise est retourné en texte brut. Par exemple pour la balise '<img src="test.jpg" alt="image">mon image</img>', le texte brut est "mon image" (fonctionnement par défaut dans les formulaires)
ST Expressions 4D comme sources	Entier long	2	La chaîne d'origine des références d'expressions 4D est retournée
ST Expressions 4D comme valeurs	Entier long	1	Les références d'expressions 4D sont retournées sous leur forme évaluée (fonctionnement par défaut dans les formulaires)
ST Liens utilisateur comme libellés	Entier long	16	Le libellé visible du lien utilisateur est retourné (fonctionnement par défaut dans les formulaires)
ST Liens utilisateur comme liens	Entier long	32	Le contenu du lien utilisateur est retourné
ST Références comme espaces	Entier long	0	Chaque référence est retournée sous forme d'un caractère espace insécable (fonctionnement par défaut, utilisé par les autres commandes)
ST Texte visible avec Expressions 4D comme sources	Entier long	86	Retourne le texte tel qu'il est visible dans les formulaires avec la chaîne d'origine des expressions 4D. Correspond à la combinaison prédéfinie des constantes 2+4+16+64.
ST Texte visible avec Expressions 4D comme valeurs	Entier long	85	Retourne le texte tel qu'il est visible dans les formulaires avec les expressions 4D sous leur forme évaluée. Correspond à la combinaison prédéfinie de constantes 1+4+16+64.
ST URL comme libellés	Entier long	4	Le libellé visible des URLs est retourné, par exemple "Visitez notre site Web" (fonctionnement par défaut dans les formulaires)
ST URL comme liens	Entier long	8	Le lien est retourné, par exemple "http://www.4d.com"

Note : Le paramètre optionnel *modeRéf* n'est donc utile que si le texte contient des références, sinon le texte brut est identique quelle que soit la valeur du paramètre *modeRéf*.

Exemple 1

Vous cherchez le texte "très beau" parmi les valeurs d'un champ texte multistyle. La valeur a été stockée sous la forme "Il fait très beau **aujourd'hui**".

```
CHERCHER PAR FORMULE([Commentaires];ST Lire texte brut([Commentaires]Meteo)="@très beau@")
```

Note : Dans ce contexte, l'instruction suivante ne donnera pas le résultat escompté car le texte est enregistré avec des balises de style :

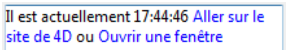
```
CHERCHER([Commentaires];[Commentaires]Meteo="@très beau@")
```

Exemple 2

Soit le texte suivant placé dans la zone multistyle "mazone" :

```
<span>Il est actuellement <span style="-d4-ref:'Heure courante:C178'"> </span><a href="http://www.4d.com"> Aller
sur le site de 4D</a> ou <span style="-d4-ref-user:'openW'">Ouvrir une fenêtre</span></span></a></span>
```

Ce texte est affiché :



Si vous exécutez le code suivant :

```
$txt :=ST Lire texte brut(*;"mazone";ST Références comme espaces)
    //$txt = "Il est actuellement    ou " (espaces)
$txt :=ST Lire texte brut(*;"mazone";ST Expressions 4D comme valeurs)
    //$txt = "Il est actuellement 18:29:55  ou  "
$txt :=ST Lire texte brut(*;"mazone";ST Expressions 4D comme sources)
    //$txt = "Il est actuellement Heure courante  ou  "
$txt :=ST Lire texte brut(*;"mazone";ST URL comme liens)
    //$txt = "Il est actuellement  http://www.4d.com ou  "
$txt :=ST Lire texte brut(*;"mazone";ST Texte visible avec Expressions 4D comme valeurs)
    //$txt = "Il est actuellement 17:54:30 Aller sur le site de 4D ou Ouvrir une fenêtre"
$txt :=ST Lire texte brut(*;"mazone";ST Texte visible avec Expressions 4D comme sources)
    //$txt = "Il est actuellement Heure courante Aller sur le site de 4D ou Ouvrir une fenêtre"
$txt :=ST Lire texte brut(*;"mazone";ST Liens utilisateur comme libellés)
    //$txt = "Il est actuellement    ou Ouvrir une fenêtre"
$txt :=ST Lire texte brut(*;"mazone";ST Liens utilisateur comme liens)
    //$txt = "Il est actuellement    ou openW"
```

Variables et ensembles système

A l'issue de l'exécution de cette commande, la variable OK prend la valeur 1 si aucune erreur n'a été rencontrée et 0 dans le cas contraire. C'est le cas notamment lorsque l'évaluation des balises de style échoue (balise incorrecte ou manquante).

A noter qu'en cas d'erreur, la variable n'est pas modifiée. Lorsqu'une erreur se produit sur une variable lors de l'évaluation du texte, 4D transforme le texte en texte brut ; par conséquent, les caractères <, > et & seront convertis en entités HTML.

ST Lire type contenu

ST Lire type contenu ({* ;} objet {; débutSél {; finSél {; débutBloc {; finBloc}}}}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
débutSél	Entier long	⇒	Début de la sélection
finSél	Entier long	⇒	Fin de la sélection
débutBloc	Entier long	⇐	Début de position du premier type de la sélection
finBloc	Entier long	⇐	Fin de position du premier type de la sélection
Résultat	Entier long	⇐	Type de contenu

Description

La commande **ST Lire type contenu** retourne le type de contenu présent dans le champ ou la variable de texte multistyle désigné(e) par le paramètre *objet*.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande retourne les informations de l'objet en cours d'édition ; si l'objet n'a pas le focus, la commande retourne les informations de la source de données de l'objet (variable ou champ).
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. A l'exécution, la commande retourne les informations de la variable ou du champ.

Les paramètres optionnels *débutSél* et *finSél* permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style éventuellement présentes dans le texte.

- Si vous passez *débutSél* et *finSél*, **ST Lire type contenu** évalue le contenu situé à l'intérieur de cette sélection.
- Si vous passez uniquement *débutSél* ou si la valeur de *finSél* est supérieure au nombre total de caractères dans l'*objet*, le contenu situé entre *débutSél* et la fin du texte est évalué.
- Si vous omettez *débutSél* et *finSél*, le contenu situé à l'intérieur de la sélection courante de texte est évalué.

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Note : Si *débutSél* est supérieur à *finSél* (hormis si *finSél* vaut 0), la commande ne fait rien et la variable *OK* prend la valeur 0.

Les paramètres optionnels *débutBloc* et *finBloc* permettent de récupérer la position du premier et du dernier caractère du premier bloc homogène identifié dans l'objet ou la sélection de l'objet. Par exemple, si la sélection contient une expression puis du texte brut, *débutBloc* et *finBloc* retourneront les bornes de l'expression. Vous pouvez effectuer une boucle afin de traiter tous les blocs de la sélection.

La commande retourne une valeur désignant le type de contenu identifié. Vous pouvez comparer cette valeur aux constantes suivantes, placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Type balise inconnu	Entier long	4	La sélection contient uniquement une balise de type inconnu
ST Type brut	Entier long	0	La sélection contient du texte et aucune référence
ST Type expression	Entier long	2	La sélection contient uniquement une référence d'expression
ST Type image	Entier long	6	La sélection contient uniquement une image (zones 4D Write Pro uniquement)
ST Type mixte	Entier long	3	La sélection contient au moins deux types de contenus différents
ST Type Url	Entier long	1	La sélection contient uniquement une référence d'URL
ST Type utilisateur	Entier long	5	La sélection contient uniquement une référence personnalisée

Exemple

Vous souhaitez afficher des commandes d'un menu contextuel en fonction du type de contenu sélectionné dans la zone.

```
Au cas ou
   :(Evenement formulaire=Sur clic)
  //on récupère la sélection
      TEXTE SELECTIONNE(*;"myText";startSel;endSel)
      Si(Clic contextuel & (Macintosh control enfoncee=Faux)) //appel du menu contextuel
         Si(startSel=endSel) //pas de contenu sélectionné
  //on active uniquement certaines commandes
            INACTIVER LIGNE MENU(<>menu_STYLEDTEXT;2)
            INACTIVER LIGNE MENU(<>menu_STYLEDTEXT;4)
            ACTIVER LIGNE MENU(<>menu_STYLEDTEXT;6)
            ...
         Sinon //on lit le type de contenu
            CT_Texttype:=ST Lire type contenu(*;"myText";startSel;endSel)
            Au cas ou //traitement des différents types
```

```
                :(CT_Texttype=ST Type Url)
                   INACTIVER LIGNE MENU(<>menu_STYLEDTEXT;6)
                   ACTIVER LIGNE MENU(<>menu_STYLEDTEXT;7)
                   ...
                :(CT_Texttype=ST Type expression)
                   INACTIVER LIGNE MENU(<>menu_STYLEDTEXT;6)
                   INACTIVER LIGNE MENU(<>menu_STYLEDTEXT;7)
                   ...
                Sinon
                   ACTIVER LIGNE MENU(<>menu_STYLEDTEXT;6)
                   INACTIVER LIGNE MENU(<>menu_STYLEDTEXT;7)
                   ...
             Fin de cas
          Fin de si
          POSITION SOURIS($xCoord;$yCoord;$StateButton)
          $AlphaVar:=Pop up menu dynamique(<>menu_STYLEDTEXT;"";$xCoord;$yCoord)
          startSel:=-3
          endSel:=-3
       Fin de si
       ...
Fin de cas
```

ST LIRE URL

ST LIRE URL ({* ;} objet ; texteURL ; adresseURL {; débutSél {; finSél}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
texteURL	Texte	⇐	Libellé visible de l'URL
adresseURL	Texte	⇐	Adresse de l'URL
débutSél	Entier long	⇒	Début de la sélection
finSél	Entier long	⇒	Fin de la sélection

Description

La commande **ST LIRE URL** retourne le libellé et l'adresse du premier URL détecté dans le champ ou la variable de texte multistyle désigné(e) par le paramètre *objet*.

Le libellé et l'adresse sont retournés dans les paramètres *texteURL* et *adresseURL*. Si la sélection ne contient aucun URL, des chaînes vides sont retournées dans ces paramètres.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre *objet* est un nom d'objet (une chaîne). A l'exécution, si l'objet a le focus, la commande retourne les informations de l'objet en cours d'édition ; si l'objet n'a pas le focus, la commande retourne les informations de la source de données de l'objet (variable ou champ).
Si vous omettez le paramètre *, vous indiquez que le paramètre *objet* est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable. A l'exécution, la commande retourne les informations de la variable ou du champ.

Les paramètres optionnels *débutSél* et *finSél* permettent de désigner une sélection de texte dans *objet*. Les valeurs *débutSél* et *finSél* expriment une sélection de texte brut, sans tenir compte des balises de style éventuellement présentes dans le texte.

- Si vous passez *débutSél* et *finSél*, **ST LIRE URL** recherche l'URL à l'intérieur de cette sélection.
- Si vous passez uniquement *débutSél* ou si la valeur de *finSél* est supérieure au nombre total de caractères dans l'*objet*, la commande recherche l'URL entre *débutSél* et la fin du texte.
- Si vous omettez *débutSél* et *finSél*, la commande recherche l'URL à l'intérieur de la sélection courante de texte.

4D propose des constantes prédéfinies afin de désigner automatiquement des bornes de sélection dans les paramètres *débutSél* et *finSél*. Ces constantes sont placées dans le thème "**Texte multistyle**" :

Constante	Type	Valeur	Comment
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet

(*) Vous devez passer un nom d'objet dans *objet* pour pouvoir utiliser cette constante. Si vous passez une référence de variable ou de champ, la commande s'appliquera à l'ensemble du texte de l'objet.

Note : Si *débutSél* est supérieur à *finSél* (hormis si *finSél* vaut 0), la commande ne fait rien et la variable *OK* prend la valeur 0.

Exemple

Sur un événement double-clic, vous vérifiez que vous êtes bien en présence d'un URL, et dans ce cas affichez un dialogue où vous avez récupéré ses valeurs afin de permettre à l'utilisateur de le modifier :

```
Au cas ou
   :(Evenement formulaire=Sur double clic)
      TEXTE SELECTIONNE(*;"StyledText_t";startSel;endSel)
      Si(ST Lire type contenu(*;"StyledText_t";startSel;endSel)=ST Type Url)    //URL
         ST LIRE URL(*;"StyledText_t";vTitle;vURL;startSel;endSel)
         $winRef:=Creer fenetre formulaire("Dial_InsertURL";Form dialogue modal déplaçable;Centrée
horizontalement;Centrée verticalement;*)
         CHANGER TITRE FENETRE("URL settings")
         DIALOGUE("Dial_InsertURL")
         Si(OK=1)
            ST INSERER URL(*;"StyledText_t";vTitle;vURL;startSel;endSel)
            SELECTIONNER TEXTE(*;"StyledText_t";startSel;startSel+1)
         Fin de si
      Fin de si
Fin de cas
```

Transactions

- Utiliser des transactions
- Suspendre des transactions
- ANNULER TRANSACTION
- DEBUT TRANSACTION
- Niveau de la transaction
- REACTIVER TRANSACTION
- SUSPENDRE TRANSACTION
- Transaction active
- Transaction en cours
- VALIDER TRANSACTION

Utiliser des transactions

Les transactions sont une série de modifications effectuées à l'intérieur d'un process sur des données reliées entre elles. Une transaction n'est sauvegardée de façon définitive dans la base que si la transaction est validée. Si une transaction n'est pas complétée, parce qu'elle est annulée ou en raison d'un quelconque événement extérieur, les modifications ne sont pas sauvegardées.

Pendant une transaction, toutes les modifications effectuées sur les données de la base dans le process sont stockées localement dans un buffer temporaire. Si la transaction est acceptée avec **VALIDER TRANSACTION**, les changements sont sauvegardés de façon définitive. Si la transaction est annulée avec **ANNULER TRANSACTION**, les changements ne sont pas sauvegardés. Dans tous les cas, ni la sélection courante ni l'enregistrement courant ne sont modifiés par les commandes de gestion des transactions.

4D prend en charge les transactions imbriquées, c'est-à-dire les transactions sur plusieurs niveaux hiérarchiques. Le nombre de sous-transactions autorisées est illimité. La commande **Niveau de la transaction** permet de connaître le niveau courant de transaction dans lequel le code est exécuté. Lorsque vous utilisez des transactions imbriquées, le résultat de chaque sous-transaction dépend de la validation ou de l'annulation de la transaction du niveau supérieur. Si la transaction supérieure est validée, les résultats des sous-transactions sont entérinés (validation ou annulation). En revanche, si la transaction supérieure est annulée, toutes les sous-transactions sont annulées, quels que soient leurs sous-résultats.

Note : Par compatibilité, les transactions imbriquées sont désactivées par défaut dans les bases de données converties depuis une version antérieure à la v11 (cf. section **Page Compatibilité**).

4D inclut une fonctionnalité vous permettant de suspendre temporairement et de réactiver des transactions dans votre code 4D. Lorsqu'une transaction est *suspendue*, vous pouvez exécuter des opérations indépendantes de la transaction elle-même puis la réactiver afin de la valider ou de l'annuler, de façon classique. Lorsque la transaction est suspendue, vous pouvez, en particulier :

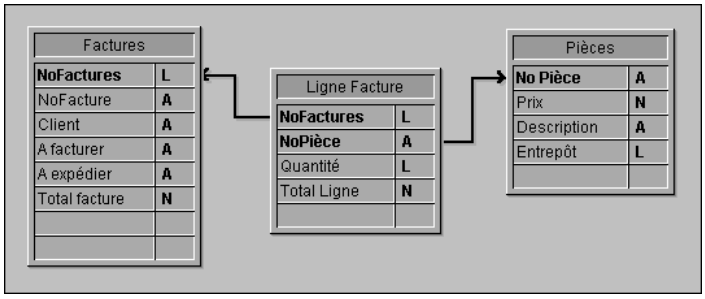
- créer ou modifier des enregistrements en-dehors de la transaction (par exemple pour incrémenter un compteur de numéros de factures),
- débuter, suspendre et/ou refermer d'autres transactions.

Pour plus d'informations sur ce point, reportez-vous à la section **Suspendre des transactions**.

Exemples de transactions

L'exemple de cette section s'appuie sur la structure présentée ci-dessous. C'est une base relativement simple de facturation. Les lignes de factures sont stockées dans une table appelée [Ligne Facture], qui est reliée à la table [Factures] par une relation entre les champs [Factures]NoFacture et [Ligne Facture]NoFacture. Lorsqu'une facture est ajoutée, un numéro unique est calculé avec la commande **Numerotation automatique**. Le lien entre [Factures] et [Ligne Facture] est du type aller-retour automatique. L'option "Mise à jour auto dans les sous-formulaires" est cochée.

La lien entre [Ligne Facture] et [Pièces] est manuel.



Quand un utilisateur saisit une facture, les actions suivantes doivent être exécutées :

- Ajouter un enregistrement dans la table [Factures].
- Ajouter plusieurs enregistrements dans la table [Ligne Facture].
- Mettre à jour le champ [Pièces]Entrepôt pour chaque pièce figurant sur la facture.

En d'autres termes, vous devez sauvegarder les données liées. C'est la situation type où vous devez utiliser une transaction. Vous pourrez ainsi être certain de pouvoir soit sauvegarder tous ces enregistrements pendant l'opération, soit annuler la transaction si un enregistrement ne peut être ajouté ou mis à jour.

Si vous n'utilisez pas une transaction, vous ne pouvez pas garantir l'intégrité logique des données de votre base. Par exemple, si un enregistrement parmi ceux de la table [Pièces] est verrouillé, vous ne pourrez pas mettre à jour la quantité stockée dans le champ [Pièces]Entrepôt. Ce champ sera alors logiquement incorrect. La somme des pièces vendues et restantes dans l'entrepôt ne sera pas égale à la quantité d'origine saisie dans l'enregistrement. Vous pouvez éviter cette situation en utilisant les transactions.

Il y a plusieurs façons d'effectuer une saisie sous transaction :

(1) Vous pouvez gérer les transactions en utilisant les commandes de transaction **DEBUT TRANSACTION**, **VALIDER TRANSACTION** et **ANNULER TRANSACTION**. Vous pouvez par exemple écrire :

```
LECTURE ECRITURE([Ligne Facture])
LECTURE ECRITURE([Pièces])
FORM FIXER ENTREE([Factures];"Saisie")
Repeter
   DEBUT TRANSACTION
   AJOUTER ENREGISTREMENT([Factures])
   Si(OK=1)
      VALIDER TRANSACTION
   Sinon
      ANNULER TRANSACTION
   Fin de si
Jusque(OK=0)
LECTURE SEULEMENT(*)
```

(2) Pour réduire les verrouillages des enregistrements pendant la saisie de données, vous pouvez aussi choisir de gérer les transactions à partir de la méthode du formulaire et d'accéder aux tables en **LECTURE ECRITURE** uniquement quand cela est nécessaire.
Vous effectuez la saisie de données en utilisant le formulaire de saisie pour [Factures], qui contient la table liée [Factures]Lignes dans un sous-formulaire. Le formulaire comporte deux boutons : *bAnnuler* et *bOK*. Aucune action ne leur est attribuée.

La boucle d'ajout devient alors :

```
LECTURE ECRITURE([Ligne Facture])
LECTURE SEULEMENT([Pièces])
FORM FIXER ENTREE([Factures];"Input")
Repeter
   AJOUTER ENREGISTREMENT([Factures])
Jusque(bOK=0)
LECTURE SEULEMENT([Ligne Facture])
```

Notez que la table [Pièces] est désormais en "lecture seulement" pendant la saisie de données. L'accès en lecture/écriture ne s'active que si les données sont validées.

La transaction est ouverte dans la méthode du formulaire entrée de la table [Factures] :

```
Au cas ou
   :(Evenement formulaire=Sur chargement)
      DEBUT TRANSACTION
      [Factures]NoFactures:=Numerotation automatique([Factures]NoFactures)
   Sinon
      [Factures]Total facture:=Somme([Ligne Facture]Total ligne)
Fin de cas
```

Si vous cliquez sur le bouton *bAnnuler*, la saisie et la transaction doivent être annulées. Voici la méthode objet du bouton *bAnnuler*:

```
Au cas ou
   :(Evenement formulaire=Sur clic)
      ANNULER TRANSACTION
      NE PAS VALIDER
Fin de cas
```

Si vous cliquez sur le bouton *bOK*, la saisie et la transaction doivent être acceptées. Voici la méthode objet du bouton *bOK* :

```
Au cas ou
   :(Evenement formulaire=Sur clic)
      $NbLines:=Enregistrements trouves([Ligne Facture])
      LECTURE ECRITURE([Pièces]) ` Passer en lecture/écriture pour accéder à la table [Pièces]
      DEBUT SELECTION([Ligne Facture]) ` Commencer à la première ligne
      $ValidTrans:=Vrai ` Tout devrait marcher
      Boucle($Line;1;$NbLines) ` Pour chaque ligne
         CHARGER SUR LIEN([Ligne Facture]NoPiece)
         OK:=1 ` Vous voulez continuer
         Tant que(Enregistrement verrouille([Pièces]) & (OK=1))
 ` Essayer d'obtenir l'enregistrement en lecture/écriture
            CONFIRMER("La pièce "+[Ligne Facture]NoPiece+" est utilisée. Vous attendez ?")
            Si(OK=1)
               ENDORMIR PROCESS(Numero du process courant;60)
               CHARGER ENREGISTREMENT([Pièces])
            Fin de si
         Fin tant que
         Si(OK=1)
 ` Mettre à jour quantité dans l'entrepôt
            [Pièces]Entrepôt:=[Pièces]Entrepôt-[Ligne Facture]Quantité
            STOCKER ENREGISTREMENT([Pièces]) ` Sauvegarder l'enregistrement
         Sinon
            $Ligne:=$NbLines+1 ` Sortir de la boucle
            $ValidTrans:=Faux
         Fin de si
         ENREGISTREMENT SUIVANT([Ligne Facture]) ` Aller à la ligne suivante
      Fin de boucle
      LECTURE SEULEMENT([Pièces]) ` Mettre la table en mode lecture seulement
      Si($ValidTrans)
         STOCKER ENREGISTREMENT([Factures]) ` Sauvegarder les enregistrements
         VALIDER TRANSACTION ` Valider toutes les modifications de la base
      Sinon
         ANNULER TRANSACTION ` Tout annuler
      Fin de si
      NE PAS VALIDER ` Quitter le formulaire
Fin de cas
```

Dans le code ci-dessus, quel que soit le bouton sur lequel l'utilisateur a cliqué, nous appelons la commande **NE PAS VALIDER**. Le nouvel enregistrement n'est pas validé par un appel à **VALIDER** mais par **STOCKER ENREGISTREMENT**. De plus, vous remarquez que **STOCKER ENREGISTREMENT** est appelée juste avant la commande **VALIDER TRANSACTION**. Ainsi, la sauvegarde de l'enregistrement [Factures] est partie intégrante de la transaction. Appeler la commande **VALIDER** validerait aussi l'enregistrement mais dans ce cas, la transaction serait validée avant le stockage de la facture. Autrement dit, l'enregistrement serait sauvegardé en-dehors de la transaction.

En fonction de vos besoins, personnalisez votre base à votre convenance, comme dans les exemples précédents. Dans le dernier exemple, la gestion du

verrouillage des enregistrements de la table [Pièces] pourrait être plus élaborée.

Suspendre des transactions

Principe

Suspendre une transaction est utile notamment lorsque vous devez, depuis une transaction, lancer certaines opérations qui n'ont pas besoin d'être effectuées sous le contrôle de cette transaction. Par exemple, imaginez le cas d'un client qui passe une commande, donc via une transaction, et qui en profite pour mettre à jour son adresse postale. Finalement, le client se ravise et annule sa commande. La transaction est annulée, mais pour autant vous ne souhaitez pas que la mise à jour de l'adresse le soit également. Ce cas peut typiquement être géré via la suspension de la transaction. Trois commandes permettent de gérer la suspension et la réactivation des transactions :

- **SUSPENDRE TRANSACTION** : suspend la transaction courante. Tous les enregistrements en cours de mise à jour ou de création restent verrouillés.
- **REACTIVER TRANSACTION** : réactive une transaction suspendue, le cas échéant.
- **Transaction active** : retourne **Faux** si la transaction courante est suspendue ou s'il n'y a pas de transaction courante, et **Vrai** si elle est démarrée ou réactivée.

Exemple

Cet exemple présente un cas typique où la suspension d'une transaction est utile. Dans une base de facturation (Invoices), nous voulons obtenir un nouveau numéro de facture durant une transaction. Ce numéro est calculé et stocké dans une table [Settings]. Dans un environnement multi-utilisateur, les accès doivent être protégés ; cependant, à cause de la transaction, la table [Settings] pourrait être verrouillée par un autre utilisateur alors même que ses données ne dépendent pas de la transaction principale. Dans ce cas, vous pouvez suspendre la transaction pour l'accès à la table.

```
 //Méthode standard qui crée une facture
DEBUT TRANSACTION
...
CREER ENREGISTREMENT([Invoices])
[Invoices]InvoiceID:=GetInvoiceNum //appel de la méthode pour obtenir un numéro disponible
...
STOCKER ENREGISTREMENT([Invoices])
VALIDER TRANSACTION
```

La méthode ***GetInvoiceNum*** suspend la transaction avant de s'exécuter. A noter que ce code fonctionnera même si la méthode est appelée en-dehors de toute transaction :

```
 //Méthode projet GetInvoiceNum
 //GetInvoiceNum -> prochain numéro de facture disponible
C_ENTIER LONG($0)
SUSPENDRE TRANSACTION
TOUT SELECTIONNER([Settings])
Si(Enregistrement verrouille([Settings])) //accès multi-utilisateur
   Tant que(Enregistrement verrouille([Settings]))
      MESSAGE("Attente d'enregistrement Settings verrouillé")
      ENDORMIR PROCESS(Numero du process courant;30)
      CHARGER ENREGISTREMENT([Settings])
   Fin tant que
Fin de si
[Settings]InvoiceNum:=[Settings]InvoiceNum+1
$0:=[Settings]InvoiceNum
STOCKER ENREGISTREMENT([Settings])
LIBERER ENREGISTREMENT([Settings])
REACTIVER TRANSACTION
```

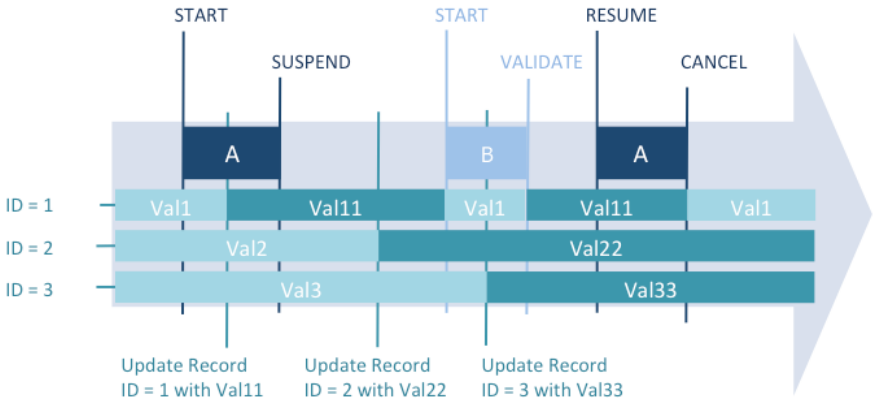
Fonctionnement détaillé

Que se passe-t-il quand une transaction est suspendue ?

Lorsqu'une transaction est suspendue, les principes de fonctionnement suivants s'appliquent :

- Vous pouvez accéder aux enregistrements qui ont été ajoutés ou modifiés durant la transaction, et vous ne pouvez pas accéder aux enregistrements qui ont été supprimés durant la transaction.
- Vous pouvez créer, sauvegarder, supprimer ou modifier des enregistrements en-dehors de la transaction.
- Vous pouvez démarrer une nouvelle transaction, mais à l'intérieur de cette transaction incluse, vous ne pourrez pas voir les enregistrements ou les valeurs d'enregistrements qui auront été modifié(s) ou ajouté(e)s dans la transaction suspendue. En fait, cette nouvelle transaction est totalement indépendante de celle qui a été suspendue, comme s'il s'agissait d'une transaction dans un autre process, et puisque la transaction suspendue pourra être par la suite validée ou annulée, tout enregistrement modifié ou annulé est automatiquement masqué pour la nouvelle transaction. Dès que vous validerez ou annulerez cette nouvelle transaction, vous pourrez à nouveau accéder à ces enregistrements.
- Tous les enregistrements modifiés, supprimés ou ajoutés à l'intérieur de la transaction suspendue restent verrouillés pour les autres process. If vous tentez de modifier ou de supprimer ces enregistrements hors de la transaction ou dans une autre transaction, une erreur est générée.

Ces principes sont résumés dans le schéma suivant :



Les valeurs modifiées durant la transaction A (enregistrement ID1 prend la valeur Val11) ne sont pas disponibles dans une nouvelle transaction (B) créée pendant la période de suspension. Les valeurs modifiées durant la période de suspension (enregistrement ID2 prend la valeur Val22 et enregistrement ID3 prend la valeur Val33) sont sauvegardées même après que la transaction A a été annulée.

Des fonctionnalités spécifiques ont été ajoutées pour prendre en charge les erreurs :

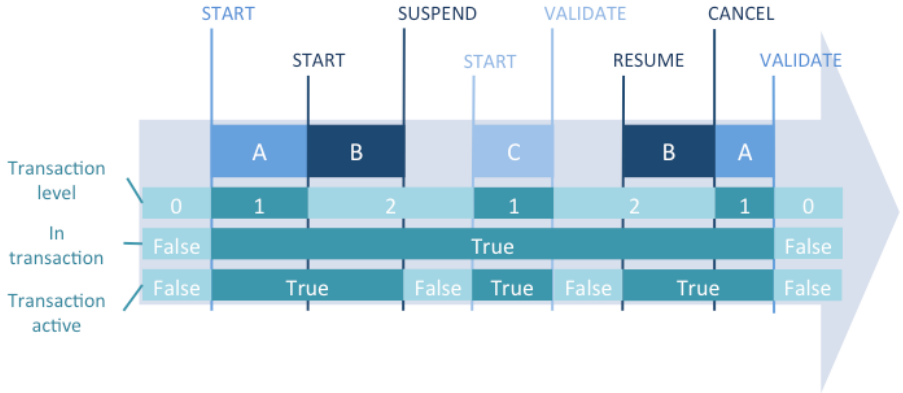
- L'enregistrement courant de chaque table devient temporairement verrouillé s'il est modifié pendant la transaction et est automatiquement déverrouillé lorsque la transaction est réactivée. Ce mécanisme est important pour empêcher que des parties de la transaction soient sauvegardées de manière impromptue.
- Si vous exécutez une séquence invalide telle que *débuter transaction / suspendre transaction / débuter transaction / réactiver transaction*, une erreur est générée. Ce mécanisme prévient tout éventuel oubli de valider ou d'annuler des sous-transactions incluses avant de réactiver une transaction suspendue.

Transactions suspendues et statut du process

La commande existante **Transaction en cours** retourne **Vrai** dès qu'une transaction a été démarrée, même si elle a été suspendue. Pour savoir si la transaction courante a été suspendue, vous devez utiliser la nouvelle commande **Transaction active**, qui retourne **Faux** dans ce cas.

Ces deux commandes, cependant, retournent également **Faux** si aucune transaction n'a été démarrée. Vous pourrez alors avoir besoin d'utiliser la commande existante **Niveau de la transaction**, qui retourne 0 dans ce contexte (pas de transaction démarrée).

Le schéma suivant illustre les différents contextes de transaction et les valeurs correspondantes retournées par les commandes de transaction :



ANNULER TRANSACTION

ANNULER TRANSACTION
Ne requiert pas de paramètre

Description

ANNULER TRANSACTION annule la transaction ouverte par la commande **DEBUT TRANSACTION** de niveau correspondant dans le process courant.
ANNULER TRANSACTION annule toutes les opérations exécutées sur les données et stockées pendant la transaction.

Note : ANNULER TRANSACTION est sans effet sur les modifications éventuellement effectuées dans les enregistrements courants mais non stockées - elles restent affichées après l'exécution de la commande.

DEBUT TRANSACTION

DEBUT TRANSACTION
Ne requiert pas de paramètre

Description

DEBUT TRANSACTION débute une transaction dans le process courant. Toutes les modifications effectuées sur les données (enregistrements) de la base à l'intérieur de la transaction seront stockées temporairement jusqu'à ce que la transaction soit validée ou annulée.

A compter de la version 11 de 4D, vous pouvez imbriquer plusieurs transactions (sous-transactions). Chaque transaction ou sous-transaction doit être finalement annulée ou validée. A noter que si la transaction principale est annulée, toutes les sous-transactions sont annulées, quels que soient leurs résultats.

Niveau de la transaction

Niveau de la transaction -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Niveau de transaction courant (0 si aucune transaction n'a été démarrée)

Description

La commande **Niveau de la transaction** retourne le niveau de transaction courant pour le process. Cette commande prend en compte toutes les transactions du process courant, qu'elles aient été démarrées via le langage de 4D ou via le SQL.

REACTIVER TRANSACTION

REACTIVER TRANSACTION
Ne requiert pas de paramètre

Description

La commande **REACTIVER TRANSACTION** réactive la transaction qui a été suspendue à l'aide de la commande **SUSPENDRE TRANSACTION** au niveau correspondant dans le process courant. Toute opération effectuée après l'appel de cette commande retourne sous le contrôle de la transaction (hormis si plusieurs transactions suspendues sont imbriquées).

Pour plus d'informations, veuillez vous référer à la section **Suspendre des transactions**.

SUSPENDRE TRANSACTION

SUSPENDRE TRANSACTION
Ne requiert pas de paramètre

Description

La commande **SUSPENDRE TRANSACTION** suspend les mécanismes de la transaction courante dans le process courant. Vous pouvez alors manipuler des données dans d'autres parties de la base, sans qu'elles soient contrôlées par la transaction, tout en préservant le contexte courant de la transaction. Tout enregistrement qui a été mis à jour ou ajouté durant la transaction est verrouillé jusqu'à ce que la transaction soit réactivée à l'aide de la commande **REACTIVER TRANSACTION**.

Pour plus d'informations, veuillez vous référer à la section **Suspendre des transactions**.

Transaction active

Transaction active -> Résultat

Paramètre	Type	Description
Résultat	Booléen	Faux si la transaction courante est suspendue

Description

La commande **Transaction active** retourne **Vrai** si le process courant est en transaction et si cette transaction n'est pas suspendue. Elle retourne **Faux** s'il n'y a pas de transaction en cours, ou si la transaction en cours est suspendue. Une transaction peut être suspendue à l'aide de la commande **SUSPENDRE TRANSACTION**.

Comme cette commande retourne également **Faux** lorsque le process courant n'est pas en transaction, vous aurez besoin d'utiliser la commande **Transaction en cours** afin de vérifier que le process est bien en transaction.

Pour plus d'informations, reportez-vous à la section **Suspendre des transactions**.

Description

Vous voulez connaître le statut courant de transaction :

```
Si(Transaction en cours)
   Si(Non(Transaction active))
      ALERTE("La transaction courante est suspendue")
   Sinon
      ALERTE("La transaction courante est active")
   Fin de si
Sinon
   ALERTE("Nous ne sommes pas en transaction")
Fin de si
```

Transaction en cours

Transaction en cours -> Résultat

Paramètre	Type		Description
Résultat	Booléen		VRAI si le process courant est en transaction, FAUX sinon

Description

La commande **Transaction en cours** retourne **Vrai** si le process courant est en transaction, sinon elle retourne **Faux**.

Exemple

Si vous effectuez des opérations (ajout, modification ou suppression) sur de multiples enregistrements, vous pouvez rencontrer des enregistrements verrouillés. Dans ce cas, pour préserver l'intégrité des données, vous devez avoir ouvert une transaction, de manière à ce que vous puissiez faire “marche arrière” et annuler l'ensemble de l'opération depuis le début, sans que les données de la base soient modifiées.

Si vous effectuez l'opération depuis un trigger ou une sous-routine pouvant être appelé(e) dans une transaction ou hors transaction, l'utilisation de la commande **Transaction en cours** vous permet de vérifier que la méthode du process courant ou la méthode appelante a bien ouvert une transaction. Si ce n'est pas le cas, vous ne commencez même pas l'opération, car, en cas d'échec au cours du processus, vous ne pourriez pas revenir sur les opérations déjà effectuées.

VALIDER TRANSACTION

VALIDER TRANSACTION
Ne requiert pas de paramètre

Description

VALIDER TRANSACTION accepte la transaction ouverte par la commande **DEBUT TRANSACTION** de niveau correspondant dans le process courant.
VALIDER TRANSACTION sauvegarde toutes les modifications effectuées sur les données de la base pendant la transaction.

A compter de la version 11 de 4D, vous pouvez imbriquer plusieurs transactions (sous-transactions). Si la transaction principale est annulée, toutes les sous-transactions sont annulées, même si elles ont été validées individuellement à l'aide de cette commande.

Variables et ensembles système

La variable système OK prend la valeur 1 si la transaction a été correctement validée, sinon elle prend la valeur 0.

A noter que lorsque OK vaut 0, la transaction est automatiquement annulée en interne (équivaut à un **ANNULER TRANSACTION**). Par conséquent, notamment dans le contexte de transactions imbriquées, il ne faut pas appeler explicitement **ANNULER TRANSACTION** si OK=0 car l'annulation sera alors appliquée à la transaction du niveau supérieur.

Triggers

Présentation des triggers

Un trigger est une méthode associée à une table. C'est une propriété d'une table. Vous n'appelez pas un trigger, les triggers sont appelés automatiquement par le moteur de 4D à chaque fois qu'un enregistrement de la table est manipulé (ajout, suppression et modification). Les triggers sont des méthodes qui peuvent éviter des opérations “illégales” dans votre base. Par exemple, dans une facturation, vous pouvez empêcher qu'un utilisateur crée une facture sans spécifier à qui elle doit être adressée. Les triggers sont un outil puissant permettant de contrôler les opérations sur les tables, et d'éviter des pertes de données accidentelles. Vous pouvez créer des triggers très simples et les rendre de plus en plus sophistiqués.

Activer et créer un trigger

Par défaut, lorsque vous créez une table en mode Développement, la table n'a pas de trigger.

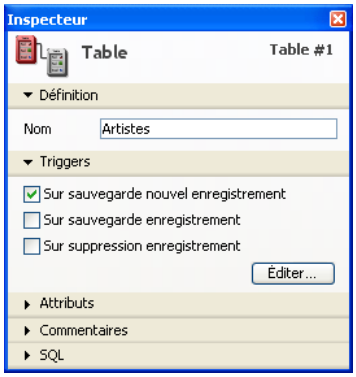
Pour utiliser un trigger pour une table, vous devez :

- activer le trigger et indiquer à 4D quand l'appeler.
- créer et écrire le code pour le trigger.

Activer un trigger qui n'est pas encore écrit ou écrire un trigger sans l'activer n'affecte pas les opérations effectuées sur une table.

1. Activer un Trigger

Pour activer le trigger, sélectionnez les options **Triggers** pour la table dans la fenêtre de l'Inspecteur de structure :



Voici la description de ces options :

Sur sauvegarde enregistrement

Si cette option est sélectionnée, le trigger sera appelé à chaque fois qu'un enregistrement de la table est modifié, c'est-à-dire dans les circonstances suivantes :

- Vous modifiez un enregistrement en saisie de données (mode Développement, commande **MODIFIER ENREGISTREMENT** ou commande SQL **UPDATE**)
- Vous sauvegardez un enregistrement existant avec **STOCKER ENREGISTREMENT**.
- Vous appelez une commande qui provoque la sauvegarde d'un enregistrement existant (par exemple, **TABLEAU VERS SELECTION**, **APPLIQUER A SELECTION**, etc.)
- Vous utilisez un plug-in 4D qui appelle la commande **STOCKER ENREGISTREMENT**.

Note : Pour des raisons d'optimisation, le trigger n'est pas appelé lors de la sauvegarde de l'enregistrement par l'utilisateur ou via la commande **STOCKER ENREGISTREMENT** si aucun champ de la table n'a été modifié dans l'enregistrement. Si vous souhaitez "forcer" l'appel du trigger dans ce cas, il suffit d'auto-affecter un champ :

```
[latable]lechamp:=[latable]lechamp
```

Sur suppression enregistrement

Si cette option est sélectionnée, le trigger sera appelé à chaque fois qu'un enregistrement de la table est supprimé, c'est-à-dire dans les circonstances suivantes :

- Vous supprimez un enregistrement en mode Développement ou en appelant la commande **SUPPRIMER ENREGISTREMENT**, **SUPPRIMER SELECTION** ou la commande SQL **DELETE**.
- Vous effectuez des opérations qui provoquent la suppression d'un enregistrement lié par l'intermédiaire des options de contrôle de suppression d'un lien.
- Vous utilisez un plug-in 4D qui appelle la commande **SUPPRIMER ENREGISTREMENT**.

Note : La commande **VIDER TABLE** ne provoque PAS l'appel du trigger.

Sur sauvegarde nouvel enregistrement

Si cette option est sélectionnée, le trigger sera appelé à chaque fois qu'un enregistrement est créé dans la table, c'est-à-dire dans les circonstances suivantes :

- Vous ajoutez un enregistrement lors de la saisie de données (mode Développement, commande **AJOUTER ENREGISTREMENT** ou commande SQL **INSERT**).
- Vous créez puis sauvegardez un enregistrement avec **CREER ENREGISTREMENT** et **STOCKER ENREGISTREMENT**. Notez que le trigger est appelé au moment où vous exécutez **STOCKER ENREGISTREMENT**, et non quand il est réellement créé.
- Vous importez des enregistrements (mode Développement ou commandes du langage).
- Vous appelez d'autres commandes qui créent et/ou sauvegardent de nouveaux enregistrements (par exemple **TABLEAU VERS SELECTION**,

STOCKER SUR LIEN, etc.).
- Vous utilisez un plug-in 4D qui appelle les commandes CREER ENREGISTREMENT et STOCKER ENREGISTREMENT.

2. Créer un trigger

Pour créer un trigger pour une table, utilisez la fenêtre de l'Explorateur, cliquez sur le bouton **Editer** dans la palette de l'Inspecteur de structure ou double-cliquez sur le titre de la table dans la fenêtre de Structure en appuyant sur la touche **Alt** (sous Windows) ou **Option** (sous Mac OS). Pour plus d'informations, reportez-vous au manuel Mode Développement.

Evénements moteur

Un trigger peut être invoqué pour l'un des trois événements moteur décrits ci-dessus. Dans le trigger, vous détectez quel événement a lieu en appelant la fonction Evenement trigger. Cette fonction retourne une valeur numérique qui indique l'événement moteur.

Typiquement, vous écrivez un trigger avec une structure du type "Au cas ou" sur le résultat retourné par Evenement trigger :

```
 // Trigger pour [UneTable]
C_ENTIER LONG($0)
$0:=0 // On suppose que la requête est acceptée
Au cas ou
   :(Evenement trigger=Sur sauvegarde nouvel enreg)
 // Effectuer les actions appropriées pour sauvegarder l'enregistrement nouvellement créé.
   :(Evenement trigger=Sur sauvegarde enregistrement)
 // Effectuer les actions appropriées pour sauvegarder l'enregistrement déjà existant.
   :(Evenement trigger=Sur suppression enregistrement)
 // Effectuer les actions appropriées pour détruire l'enregistrement.
Fin de cas
```

Les triggers sont des fonctions

Un trigger a deux finalités :

- Effectuer des actions sur l'enregistrement juste avant qu'il soit sauvegardé ou supprimé.
- Accepter ou rejeter une opération de base de données.

1. Effectuer des actions

A chaque fois qu'un enregistrement est sauvegardé (ajouté ou modifié) dans une table *[Documents]*, vous souhaitez “estampiller ” l'enregistrement avec des marqueurs de création et de modification. Vous pouvez écrire le trigger suivant :

```
 // Trigger pour table [Documents]
Au cas ou
   :(Evenement trigger=Sur sauvegarde nouvel enreg)
      [Documents]Creation Stamp:=Time stamp
      [Documents]Modification Stamp:=Time stamp
   :(Evenement trigger=Sur sauvegarde enregistrement)
      [Documents]Modification Stamp:=Time stamp
Fin de cas
```

Note : La fonction *Time stamp* utilisée dans cet exemple est une petite méthode projet retournant le nombre de secondes écoulées depuis une date choisie arbitrairement.

Une fois que ce trigger a été écrit et activé, peu importe la façon dont vous ajoutez ou modifiez un enregistrement dans la table *[Documents]* (saisie de données, import, méthode projet, plug-in 4D), la valeur des deux champs *[Documents]Creation Stamp* et *[Documents]Modification Stamp* sera automatiquement affectée par le trigger avant que l'enregistrement ne soit écrit sur disque.

Note : Voir l'exemple de la commande PROPRIETES DOCUMENT pour une analyse complète de cet exemple.

2. Accepter ou rejeter l'opération de la base

Pour accepter ou rejeter une opération de la base, le trigger doit retourner un **code d'erreur de trigger** dans le résultat de la fonction *$0*.

Exemple

Prenons le cas d'une table *[Employés]*. Pendant la saisie de données, vous contrôlez le champ *[Employés]No Séc.Soc*. Par exemple, lorsque l'utilisateur clique sur le bouton de validation, vous vérifiez le champ utilisant la méthode objet du bouton :

```
 // Méthode objet bouton bAccept
Si(Bon No Séc.Soc([Employés]No Séc.Soc))
   VALIDER
Sinon
   BEEP
   ALERTE("Saisissez un numéro de sécurité sociale et cliquez de nouveau sur OK.")
Fin de si
```

Si la valeur du champ est correcte, vous acceptez la saisie de données, sinon vous affichez une alerte et restez en saisie de données.

Si vous créez aussi des enregistrements pour la table *[Employés]* par programmation, le code ci-dessous serait valide MAIS violerait la règle imposée dans la méthode objet créée plus haut :

```
 ` Extrait d'une méthode projet
 ` ...
CREER ENREGISTREMENT([Employés])
[Employés]Nom:="DOE"
STOCKER ENREGISTREMENT([Employés]) ` <- violation de la règle ! Il n'y a pas de numéro de sécurité sociale !
```

En utilisant un trigger pour la table *[Employés]*, vous pouvez appliquer la contrainte sur *[Employés]No Séc.Soc* à tous les niveaux de la base. Le trigger serait du type :

```
 // Trigger pour [Employés]
$0:=0
$dbEvent:=Evenement trigger
Au cas ou
   :(($dbEvent=Sur sauvegarde nouvel enreg)|($dbEvent=Sur sauvegarde enregistrement))
      Si(Non(Bon No Séc.Soc([Employés]No Séc.Soc)))
         $0:=-15050
      Sinon
 // ...
      Fin de si
 // ...
Fin de cas
```

Une fois que ce trigger est écrit et activé, la ligne **STOCKER ENREGISTREMENT([Employés])** de la méthode projet ci-dessus génèrera une erreur moteur -15050 et l'enregistrement ne sera PAS sauvegardé.

De la même façon, si un plug-in 4D essayait de sauvegarder un enregistrement dans *[Employés]* avec un numéro de sécurité sociale incorrect, le trigger génèrerait la même erreur et l'enregistrement ne serait pas sauvegardé non plus.

Le trigger garantit que personne (utilisateur, développeur, plug-in...) ne peut violer la règle sur le numéro de sécurité sociale (à dessein ou par erreur).

Notez que même si vous n'avez pas créé de trigger pour une table, la base peut retourner des erreurs moteur lorsque vous essayez de sauvegarder ou de détruire un enregistrement. Vous pouvez, par exemple, recevoir l'erreur -9998, si vous essayez de sauvegarder un enregistrement.

Les triggers retournent de nouveaux types d'erreurs dans 4D :

- 4D gère les erreurs “normales” : index unique, contrôles relationnels, etc.
- En utilisant les triggers, vous pouvez créer des codes d'erreurs propres au contenu de votre application.

Important : Vous pouvez retourner le code d'erreur de votre choix. Cependant, n'utilisez pas des codes d'erreurs déjà utilisés par le moteur de 4D. Nous vous recommandons fortement d'utiliser des codes compris entre -32000 et -15000. Nous réservons les erreurs supérieures à -15000 au moteur de 4D.

Au niveau du process, vous gérez les erreurs trigger de la même façon que les erreurs du moteur de base de données :

- vous pouvez laisser 4D afficher la boîte de dialogue standard d'erreur, la méthode est alors interrompue.
- vous pouvez utiliser une méthode de gestion d'erreur installée par **APPELER SUR ERREUR** et traiter l'erreur de façon appropriée.

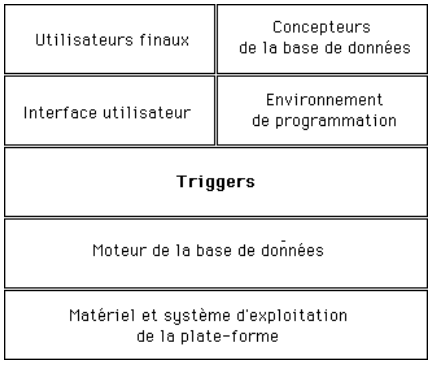
Notes :

- Pendant la saisie, si une erreur trigger est retournée au moment où vous essayez de valider ou de supprimer un enregistrement, l'erreur est gérée comme une erreur sur un index unique. La boîte de dialogue d'erreur est affichée et vous restez en saisie de données. Même si vous n'utilisez une base qu'en mode Développement (et non en Application), vous bénéficiez des triggers.
- Lorsqu'une erreur est générée par un trigger dans le cadre d'une commande agissant sur une sélection d'enregistrements (telle que **SUPPRIMER SELECTION**), l'exécution de la commande est immédiatement stoppée, sans que la sélection ait été nécessairement traitée en totalité. Ce cas requiert de la part du développeur une gestion appropriée, basée par exemple sur la conservation temporaire de la sélection, le traitement et l'élimination de l'erreur avant l'exécution du trigger, etc.

Même si un trigger ne retourne pas d'erreur (*$0:=0*), cela ne signifie pas qu'une opération de la base s'effectuera correctement. Il peut y avoir eu un doublon sur l'index unique. Si l'opération est la mise à jour d'un enregistrement, ce dernier peut être verrouillé, une erreur d'entrée/sortie peut se produire, bien d'autres choses encore peuvent arriver. Ces vérifications sont effectuées après l'exécution du trigger. Cependant, du point de vue du plus haut niveau du process en exécution, les erreurs retournées par le moteur de la base de données ou celle d'un trigger sont de même nature : une erreur trigger est une erreur du moteur de la base de données.

Les triggers et l'architecture 4D

Les triggers sont exécutés au niveau du moteur de la base de données. Ce point est illustré dans le schéma suivant :



Les triggers sont exécutés **sur la machine où est situé le moteur de la base de données**. Si ce point est une évidence dans le cas de 4D en local, il convient de rappeler que pour 4D Server, les triggers sont exécutés sur la machine serveur (dans le process "jumeau" du process ayant déclenché le trigger) et non sur la machine cliente.

Quand un trigger est appelé, il s'exécute dans le contexte du process qui tente l'opération. Ci-dessous, ce process est appelé **process appelant** l'exécution du trigger.
Les éléments inclus dans ce contexte diffèrent suivant que la base est exécutée avec 4D en mode local ou avec 4D Server :

- avec 4D en mode local, le trigger fonctionne avec les sélections courantes, les enregistrements courants, les statuts lecture/écriture des tables, les verrouillages d'enregistrements, etc., du process appelant.
- avec 4D Server, seul le contexte de base de données du process client appelant est préservé (verrouillages d'enregistrements et statut transactionnel). 4D Server garantit également (et uniquement) que l'enregistrement courant de la table du trigger est correctement positionné. Les

autres éléments contextuels (sélections courantes par exemple) sont ceux du process du trigger.

Soyez prudent lorsque vous utilisez les autres objets de la base et du langage, car un trigger peut s'exécuter sur une machine différente de celle du process appelant : c'est le cas avec 4D Server !

- **Variables interprocess** : Un trigger a accès aux variables interprocess de la machine où il est exécuté. Avec 4D Server, ce peut être une machine différente de celle du process appelant.
- **Variables process** : Chaque trigger possède sa propre table de variables process. Un trigger n'a pas accès aux variables process du process appelant.
- **Variables locales** : Vous pouvez utiliser des variables locales dans un trigger. Leur aire d'action est l'exécution du trigger (elles sont créées/détruites au cours de cette exécution).
- **Sémaphores** : Un trigger peut tester ou placer des sémaphores globaux et locaux (sur la machine où il s'exécute dans ce dernier cas). Cependant, un trigger doit s'exécuter rapidement. En conséquence, utilisez plutôt des sémaphores locaux dans un trigger, sauf si vous avez une idée précise en tête.
- **Ensembles et sélections temporaires** : Si vous utilisez un ensemble ou une sélection temporaire dans un trigger, vous travaillez alors avec ceux de la machine où les triggers s'exécutent. En client/serveur, les ensembles et sélections temporaires "process" (dont le nom ne débute ni par $ ni par <>) créés sur le client sont visibles dans un trigger.
- **Interface utilisateur** : N'utilisez PAS d'éléments d'interface utilisateur dans un trigger (alerte, message ou dialogue). Cela signifie également que tracer le trigger dans la fenêtre du **Débogueur** doit être limité. Souvenez-vous que les triggers en client/serveur s'exécutent sur la machine 4D Server. Un message d'alerte affiché sur le poste serveur ne dit pas grand chose à l'utilisateur qui, lui, travaille sur sa machine cliente. Laissez le process appelant gérer l'interface utilisateur.

A noter que si vous utilisez le système de mots de passe de 4D, vous pouvez exécuter la commande **Utilisateur courant** dans le trigger afin, par exemple, de stocker dans une table journal le nom de l'utilisateur à l'origine de l'appel du trigger, y compris en mode client-serveur.

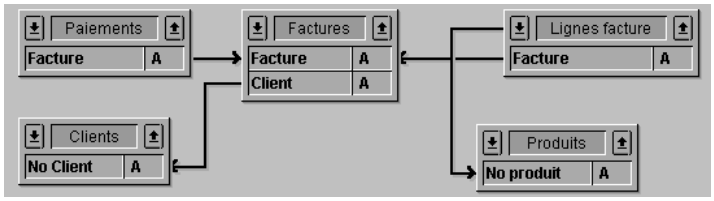
Triggers et transactions

Les transactions doivent être gérées au niveau du process appelant. Il est fortement déconseillé de gérer des transactions au niveau du trigger. Si, pendant l'exécution d'un trigger, vous devez ajouter, modifier ou détruire plusieurs enregistrements et souhaitez garantir l'intégrité de vos données à l'aide d'une transaction, vous devez d'abord tester (à partir du trigger) si le process appelant est en cours de transaction avec la commande **Transaction en cours**. En effet, si ce n'est pas le cas et si le trigger rencontre un enregistrement verrouillé, le process appelant n'aura aucun moyen d'annuler a posteriori les actions déjà effectuées par le trigger. Par conséquent, si vous n'êtes pas en transaction, ne commencez pas les opérations à exécuter, et retournez simplement une erreur dans *$0* afin de signaler au process appelant que l'opération de base de données doit être exécutée dans une transaction.

Note : Afin d'optimiser le fonctionnement combiné des triggers et des transactions, 4D n'appelle PAS les triggers lors d'un **VALIDER TRANSACTION**. Cela évite que les triggers soient exécutés deux fois.

Triggers en cascade

Prenons l'exemple de la structure suivante :



Note : Les tables ont été contractées (il y a davantage de champs).

Pour les nécessités de cette documentation, nous admettrons que la base “autorise” la suppression d'une facture. Voyons aussi comment une telle opération serait gérée au niveau du trigger (puisque vous pourriez aussi décider d'effectuer l'opération au niveau du process).

Afin que soit maintenue l'intégrité relationnelle des données, la suppression d'une facture requiert les actions suivantes de la part du trigger de *[Factures]* :

- Décrémenter le champ Ventes de la table *[Clients]* du montant de la facture.
- Supprimer tous les enregistrements de *[Lignes facture]* liés à la facture.
- Ceci implique aussi que le trigger de *[Lignes facture]* décrémente le champ Quantité vendue des enregistrements *[Produits]* liés à la ligne de facture que l'on s'apprête à supprimer.
- Supprimer tous les enregistrements de *[Paiements]* liés à la facture.

Tout d'abord, le trigger de *[Factures]* ne doit effectuer ces actions que si le process appelant est en transaction, afin qu'une annulation rétroactive soit possible en cas de rencontre d'un enregistrement verrouillé.

Deuxièmement, le trigger de *[Lignes facture]* est **en cascade** avec le trigger de *[Factures]*. Le premier s'exécute “à l'intérieur” du second parce que la destruction des éléments de la liste est consécutive à un appel à **SUPPRIMER SELECTION** dans le trigger de *[Factures]*.

Ajoutons que toutes les tables dans cet exemple ont des triggers activés pour tous les événements de la base de données. La cascade des triggers sera :

- Le trigger de Factures est appelé car le process appelant supprime une facture
  - Le trigger de Clients est appelé car le trigger Factures met à jour le champ Ventes
  - Le trigger de Lignes facture est appelé car le trigger Factures supprime une ligne (ce qui est répété)
    - Le trigger de Produits est appelé car le trigger Lignes facture met à jour le champ Quantité vendue
  - Le trigger de Paiements est appelé car le trigger Factures supprime un paiement (ce qui est répété)

Dans cette cascade, le trigger de [Factures] s'exécute au niveau 1, les triggers [Clients], [Lignes facture] et [Paiements] au niveau 2 et le trigger [Produits] au niveau 3.

Dans les triggers, vous pouvez détecter à quel niveau un trigger est exécuté grâce à la commande **Niveau du trigger**. De plus, vous pouvez aussi obtenir des informations sur les autres niveaux en utilisant la commande **PROPRIETES DU TRIGGER**.

Si, par exemple, vous détruisiez un enregistrement *[Produits]* à un niveau process, le trigger de *[Produits]* s'exécuterait au niveau 1, non au niveau 3, comme plus haut.

Avec **Niveau du trigger** et **PROPRIETES DU TRIGGER**, vous pouvez identifier la raison d'une action. Dans l'exemple ci-dessus, une facture est supprimée au niveau process. Prenons pour hypothèse que nous voulons détruire un enregistrement *[Clients]* au niveau process. Le trigger de *[Clients]* devrait alors être conçu pour détruire toutes les factures liées à ce client. Cela signifie que le trigger *[Factures]* devrait être invoqué comme plus haut, mais pour une autre raison. Du trigger *[Factures]*, vous pouvez détecter si le niveau est 1 ou 2. S'il est 2, vous pouvez vérifier si oui ou non c'est à cause de la suppression de l'enregistrement Client lui-même. Si tel est le cas, vous n'avez même plus besoin de vous préoccuper de la mise à jour du champ Ventes.

Evenement trigger

Evenement trigger -> Résultat

Paramètre	Type		Description
Résultat	Entier long	➲	0=Hors de tout événement de trigger, 1=Sauvegarde d'un nouvel enregistrement, 2=Sauvegarde d'un enregistrement existant, 3=Suppression d'un enregistrement

Description

La commande **Evenement trigger** est appelée dans un trigger et renvoie une valeur numérique qui indique le type de l'événement de la base, ou la raison pour laquelle le trigger a été appelé. 4D fournit les constantes prédéfinies suivantes, placées dans le thème **Evénements trigger** :

Constante	Type	Valeur
Sur sauvegarde enregistrement	Entier long	2
Sur sauvegarde nouvel enreg	Entier long	1
Sur suppression enregistrement	Entier long	3

Si, dans un trigger, vous effectuez des opérations de base de données sur plusieurs enregistrements (par exemple mise à jour de plusieurs enregistrements dans la table *[Produits]* et ajout d'enregistrement dans la table *[Factures]*), vous pouvez rencontrer des situations (comme des enregistrements verrouillés) qui empêchent le trigger d'exécuter correctement les opérations pour lesquelles il est appelé. Il vous faut alors stopper les actions de la base et retourner une erreur pour que le process appelant sache que la requête n'a pu être exécutée. Ce process doit également être en mesure d'annuler les opérations non exécutées. Autrement dit, lorsqu'une telle situation se produit, vous avez besoin de savoir dans le trigger si vous êtes en transaction avant même d'essayer de faire quoi que ce soit. Pour cela, utilisez la fonction **Transaction en cours**.

Dans 4D, il n'y a pas de limite, à part la mémoire disponible, aux appels de triggers en cascade. Pour optimiser l'exécution d'un trigger, vous pouvez écrire le code de vos triggers non seulement en fonction de l'événement de la base mais aussi du niveau de l'appel lorsque les triggers sont appelés en cascade. Par exemple, pendant l'événement trigger **Sur suppression enregistrement** pour la table *[Factures]*, vous pouvez ne pas effectuer la mise à jour du champ *[Clients]Ventes* si la suppression de l'enregistrement de la table *[Factures]* fait partie de la suppression en cascade des factures liées à l'enregistrement dans la table *[Clients]* que vous êtes en train de supprimer. Pour cela, utilisez les routines **Niveau du trigger** et **PROPRIETES DU TRIGGER**.

Exemple

Utilisez la fonction **Evenement trigger** pour structurer vos triggers comme ci-dessous :

```
 // Un trigger de la table [toute table]
C_ENTIER LONG($0)
$0:=0 // S'assurer que la requête de la base sera accordée
Au cas ou
   :(Evenement trigger=Sur sauvegarde nouvel enreg)
 // Exécuter les actions appropriées pour la sauvegarde d'un nouvel enregistrement
   :(Evenement trigger=Sur sauvegarde enregistrement)
 // Exécuter les actions appropriées pour la sauvegarde d'un enregistrement existant
   :(Evenement trigger=Sur suppression enregistrement)
 // Exécuter les actions appropriées pour la suppression d'un enregistrement
Fin de cas
```

Niveau du trigger

Niveau du trigger -> Résultat

Paramètre	Type		Description
Résultat	Entier long		Niveau d'exécution du trigger (0 si hors du cycle d'exécution du trigger)

Description

La commande **Niveau du trigger** retourne le niveau d'exécution du trigger.
Reportez-vous à la description des **Triggers en cascade**.

PROPRIETES DU TRIGGER

PROPRIETES DU TRIGGER (niveauTrigger ; evenementBase ; numTable ; numEnreg)

Paramètre	Type		Description
niveauTrigger	Entier long	⇒	Niveau d'exécution du trigger
evenementBase	Entier long	⇐	Evénement de base de données
numTable	Entier long	⇐	Numéro de la table
numEnreg	Entier long	⇐	Numéro de l'enregistrement

Description

La commande **PROPRIETES DU TRIGGER** fournit des informations sur le niveau d'exécution du trigger que vous avez passé dans *niveauTrigger*. Vous devez utiliser conjointement **PROPRIETES DU TRIGGER** et **Niveau du trigger** pour effectuer différentes actions en fonction de la cascade du trigger. Reportez-vous à la description des triggers en cascade dans la section **Présentation des triggers**.

Si vous passez un niveau d'exécution de trigger inexistant, la commande retourne 0 (zéro) dans chaque paramètre.

La nature de l'événement de base de données pour le niveau d'exécution du trigger est retournée dans *evenementBase*. Les constantes prédéfinies suivantes sont fournies dans le thème "**Evénements trigger**" :

Constante	Type	Valeur
Sur sauvegarde enregistrement	Entier long	2
Sur sauvegarde nouvel enreg	Entier long	1
Sur suppression enregistrement	Entier long	3

Le numéro de table et d'enregistrement pour l'enregistrement concerné par l'événement de base de données pour le niveau d'exécution du trigger sont retournés dans *numTable* et *numEnreg*.

Utilisateurs et groupes

- Appartient au groupe
- BLOB VERS UTILISATEURS
- CHANGER LICENCES
- CHANGER MOT DE PASSE
- CHANGER PRIVILEGES
- CHANGER UTILISATEUR COURANT
- ECRIRE ACCES PLUGIN
- Ecrire proprietes groupe
- Ecrire proprietes utilisateur
- Licence disponible
- Lire acces plugin
- LIRE LISTE GROUPE
- LIRE LISTE UTILISATEURS
- LIRE PROPRIETES GROUPE
- LIRE PROPRIETES UTILISATEUR
- Lire utilisateur par defaut
- SUPPRIMER UTILISATEUR
- Utilisateur courant
- Utilisateur supprime
- UTILISATEURS VERS BLOB
- Valider mot de passe

Appartient au groupe

Appartient au groupe (nomUtilisateur ; groupe) -> Résultat

Paramètre	Type		Description
nomUtilisateur	Chaîne	→	Nom de l'utilisateur
groupe	Chaîne	→	Nom du groupe
Résultat	Booléen	↩	Vrai = utilisateur est dans groupe Faux = utilisateur n'est pas dans groupe

Description

La fonction **Appartient au groupe** retourne Vrai si *utilisateur* appartient au *groupe*.

Exemple

L'exemple suivant recherche des factures. Si l'utilisateur courant est dans le groupe Administration, il pourra accéder aux formulaires qui affichent des informations confidentielles. Sinon, des formulaires standard sont affichés :

```
CHERCHER([Factures];[Factures]Prix>100)
Si(Appartient au groupe(Utilisateur courant;"Administration")
   FORM FIXER SORTIE([Factures];"Confidentiel_Sortie")
   FORM FIXER ENTREE([Factures];"Conf_Saisie")
Sinon
   FORM FIXER SORTIE([Factures];"Sortie_Standard")
   FORM FIXER ENTREE([Factures];"Entrée_Standard")
Fin de si
MODIFIER SELECTION([Factures];*)
```

BLOB VERS UTILISATEURS

BLOB VERS UTILISATEURS (utilisateurs)

Paramètre	Type		Description
utilisateurs	BLOB	➡	BLOB (crypté) contenant des comptes utilisateurs créés et sauvegardés par l'Administrateur

Description

La commande **BLOB VERS UTILISATEURS** remplace les comptes utilisateurs et les groupes créés par l'Administrateur dans la base de données par ceux présents dans le BLOB *utilisateurs*. Le BLOB *utilisateurs* est crypté et doit impérativement avoir été créé par la commande **UTILISATEURS VERS BLOB**.

Seuls l'Administrateur et le Super_Utilisateur de la base peuvent exécuter cette commande. Si un autre utilisateur tente de l'exécuter, la commande ne fait rien et une erreur de privilège (-9949) est générée.

Cette commande provoque le remplacement des comptes et groupes créés par l'Administrateur éventuellement existant dans la base. Si le BLOB *utilisateurs* contient des données valide, la commande effectue les opérations suivantes :

- tous les groupes et utilisateurs définis dans la base dont le numéro de référence est négatif (groupes et utilisateurs créés par l'Administrateur) sont supprimés de la structure,
- tous les groupes et utilisateurs présents dans le BLOB *utilisateurs* sont ajoutés dans la structure.

Note de compatibilité : Les fichiers d'utilisateurs et groupes (extension .4UG) créés par la commande de menu **Enregistrer les groupes & utilisateurs...** dans une version de 4D antérieure à la 2004 peuvent être chargés dans 4D version 2004 et suivantes via ces instructions :

```
DOCUMENT VERS BLOB(mondoc;blob)
BLOB VERS UTILISATEURS(blob)
```

Variables et ensembles système

Si la commande est exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

CHANGER LICENCES

CHANGER LICENCES
Ne requiert pas de paramètre

Description

La commande **CHANGER LICENCES** affiche la boîte de dialogue du Gestionnaire de licences 4D.

Cette commande est utilisable avec une application 4D monoposte uniquement et ne peut pas être appelée depuis un composant. Lorsque les mots de passe sont activés, elle peut être exécutée uniquement par le Super_Utilisateur ou l'Administrateur ; elle ne fait rien si elle est appelée par un utilisateur ne disposant pas des privilèges adéquats.

La boîte de dialogue du Gestionnaire de licences vous permet notamment d'activer des plug-ins ou le serveur Web sur le poste où elle est exécutée. Dans 4D, vous pouvez afficher cette boîte de dialogue en sélectionnant la commande **Gestionnaire de licences...** dans le menu **Aide**.

CHANGER LICENCES permet d'activer des licences et d'ajouter des numéros d'expansion dans une application monoposte compilée diffusée à vos clients. Les développeurs 4D et les administrateurs de systèmes peuvent utiliser cette commande pour diffuser une application 4D, en laissant à leurs clients le soin de saisir eux-même les numéros sans devoir leur faire parvenir une mise à jour de l'application.

Pour plus d'informations sur le fonctionnement de cette boîte de dialogue, reportez-vous à la section **Installation et activation** du Guide d'installation de 4D.

Exemple

Dans une boîte de dialogue de configuration ou de préférences personnalisée, vous placez un bouton auquel la méthode suivante est associée :

```
// Méthode objet du bouton bLicence
CHANGER LICENCES
```

Vous permettrez ainsi à un utilisateur d'activer des licences sans avoir à modifier la base de données.

CHANGER MOT DE PASSE

CHANGER MOT DE PASSE (motDePasse)

Paramètre	Type		Description
motDePasse	Chaîne	⇨	Nouveau mot de passe

Description

CHANGER MOT DE PASSE permet de changer le mot de passe de l'utilisateur courant. Cette commande remplace le mot de passe courant par le nouveau mot de passe que vous passez dans *motDePasse*.

Attention : Les mots de passe différencient les caractères majuscules et minuscules.

Exemple

L'exemple suivant permet à l'utilisateur de modifier son mot de passe :

```
CHANGER UTILISATEUR COURANT ` Afficher la boîte de dialogue des mots de passe
Si(OK=1)
   $pw1:=Demander("Saisissez le nouveau mot de passe pour "+Utilisateur courant)
 ` Le mot de passe doit comporter au moins cinq caractères
   Si(((OK=1)&($pw1#""))&(Longueur($pw1)>5))
 ` Vérifier qu'un mot de passe valide a été saisi
      $pw2:=Demander("Saisissez de nouveau le mot de passe")
      Si((OK=1)&($pw1=$pw2))
         CHANGER MOT DE PASSE($pw2) ` Modifier le mot de passe
      Fin de si
   Fin de si
Fin de si
```

CHANGER PRIVILEGES

CHANGER PRIVILEGES
Ne requiert pas de paramètre

Description

CHANGER PRIVILEGES permet de modifier le système de mots de passe. Lorsque cette commande est exécutée, la fenêtre de la boîte à outils de 4D contenant les pages Utilisateurs et Groupes est appelée pour modifier les privilèges.

Note : Cette commande ouvre une fenêtre modale. Par conséquent, vous ne devez pas l'appeler depuis une autre fenêtre modale, sinon l'ouverture de la fenêtre sera impossible et la commande ne fera rien.

Les groupes peuvent être modifiés par le Super_Utilisateur et l'Administrateur et par les propriétaires de groupe. Seuls le Super_Utilisateur et l'Administrateur peuvent modifier tous les groupes. Les propriétaires de groupe ne peuvent modifier que leur propre groupe. Des utilisateurs peuvent être ajoutés et retirés des groupes. Cette commande ne fait rien si aucun groupe n'est défini.

Le Super_Utilisateur et l'Administrateur peuvent créer des utilisateurs et les placer dans des groupes.

Exemple

L'exemple suivant affiche la fenêtre de gestion des utilisateur et des groupes :

```
CHANGER PRIVILEGES
```

CHANGER UTILISATEUR COURANT

CHANGER UTILISATEUR COURANT {(utilisateur ; motDePasse)}

Paramètre	Type		Description
utilisateur	Chaîne, Entier long	➡	Nom ou Numéro de référence unique de l'utilisateur
motDePasse	Chaîne	➡	Mot de passe (non crypté)

Description

CHANGER UTILISATEUR COURANT permet de changer l'identité de l'utilisateur courant dans la base, sans devoir la quitter. Le changement d'identité peut être effectué par l'utilisateur lui-même via la boîte de dialogue de connexion à la base (lorsque la commande est appelée sans paramètres) ou directement par la commande. Lorsqu'il change d'identité, l'utilisateur abandonne ses anciens privilèges au profit de ceux de l'utilisateur choisi.

Le nouveau compte est utilisé pour tous les process "utilisateurs" courants de la base. A noter qu'un process créé par un autre process hérite du compte utilisateur de celui-ci.

4D Server : Sur le poste serveur, les process démarrés via les **Méthode base Sur démarrage serveur**, **Méthode base Sur arrêt serveur**, etc., sont exécutés sous le compte du Super_Utilisateur.

Si la commande **CHANGER UTILISATEUR COURANT** est exécutée sans paramètres, la boîte de dialogue de connexion à la base s'affiche. L'utilisateur doit alors saisir ou sélectionner un nom et un mot de passe valides pour entrer dans la base. Le contenu de la boîte de dialogue de connexion dépend des options définies dans la page **Sécurité** des Propriétés de la base.

Note : Cette commande requiert que le système de contrôle d'accès soit activé, c'est-à-dire qu'un mot de passe soit assigné au *Super_Utilisateur*. Dans le cas contraire, **CHANGER UTILISATEUR COURANT** est sans effet et n'affichera pas la fenêtre standard de changement d'utilisateur.

Vous pouvez également passer les deux paramètres facultatifs *utilisateur* et *motDePasse* afin de spécifier par programmation le nouveau compte à utiliser. Passez dans le paramètre *utilisateur* le nom ou le numéro de référence unique (*réfUtilisateur*) du compte à utiliser. Les noms et les numéros des utilisateurs peuvent être obtenus via la commande **LIRE LISTE UTILISATEURS**.

Numéro de référence de l'utilisateur	Description de l'utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le deuxième, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le deuxième, et ainsi de suite).

Si le compte d'utilisateur désigné n'existe pas ou a été supprimé, l'erreur -9979 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**. Sinon, vous pouvez appeler la fonction **Utilisateur supprime** pour tester le compte utilisateur avant d'appeler cette commande.

Passez dans le paramètre *motDePasse* le mot de passe non crypté du compte de l'utilisateur. Si le mot de passe ne correspond pas à l'utilisateur, la commande ne fait rien et l'erreur -9978 est générée.

La commande est temporisée afin d'éviter des attaques par "force brute" (essais automatiques de multiples combinaisons de noms d'utilisateurs/mots de passe). Ainsi, au bout du quatrième appel à la commande, elle n'est exécutée qu'après 10 secondes d'attente. Cette temporisation est globale au poste de travail.

Proposer une boîte de dialogue de gestion d'accès personnalisée

La commande **CHANGER UTILISATEUR COURANT** permet de mettre en place des boîtes de dialogue personnalisées pour la saisie du nom et du mot de passe (avec règles de saisie et d'expiration) tout en bénéficiant des avantages du système intégré de contrôle des accès de 4D.
Le principe est le suivant :

1. L'entrée dans la base s'effectue directement en mode "Utilisateur par défaut", sans boîte de dialogue.

2. Dans la **Méthode base Sur ouverture**, le développeur provoque l'affichage d'une boîte de dialogue personnalisée de saisie du nom d'utilisateur et du mot de passe (à l'aide de la commande **DIALOGUE** ou **AJOUTER ENREGISTREMENT** par exemple). Tout type de traitement peut être envisagé dans la boîte de dialogue :
- Il est possible d'afficher la liste des utilisateurs de la base, comme dans la boîte de dialogue d'accès standard de 4D, à l'aide de la commande **LIRE LISTE UTILISATEURS**.
- Le champ de saisie du mot de passe peut contenir divers contrôles afin de vérifier la validité des caractères saisis (nombre minimum de caractères, unicité...).
- Pour les caractères du mot de passe saisi soient brouillés à l'écran, vous pouvez utiliser la commande **FILTRER FRAPPE CLAVIER**.
- Des règles d'expiration peuvent être appliquées au moment de la validation de la boîte de dialogue : date d'expiration, changement forcé à la première connexion, verrouillage du compte après plusieurs saisies erronées, mémorisation des mots de passe déjà utilisés...

3. Lorsque la saisie est validée, les informations requises (nom d'utilisateur et mot de passe) sont passées à la commande **CHANGER UTILISATEUR COURANT** afin d'ouvrir la base avec les privilèges du compte utilisateur.

Exemple

L'exemple suivant affiche la boîte de dialogue de connexion :

```
CHANGER UTILISATEUR COURANT
```

ECRIRE ACCES PLUGIN

ECRIRE ACCES PLUGIN (plugIn ; groupe)

Paramètre	Type		Description
plugIn	Entier long	⇒	Numéro du plug-in
groupe	Chaîne	⇒	Nom du groupe à associer au plug-in

Description

La commande **ECRIRE ACCES PLUGIN** permet de spécifier par programmation le groupe d'utilisateurs autorisé à utiliser chaque plug-in "sérialisé" installé dans la base. Cette définition permet de gérer la répartition des licences des plug-ins.

Note : Cette opération peut également être effectuée en mode Développement dans l'éditeur de groupes.

Passez dans le paramètre *plugIn* le numéro du plug-in auquel associer un groupe d'utilisateurs. Les licences de plug-ins incluent les licences Web et SOAP de 4D Client. Vous pouvez passer une des constantes suivantes du thème "**Licence disponible**" :

Constante	Type	Valeur
Licence 4D for OCI	Entier long	808465208
Licence 4D ODBC Pro	Entier long	808464946
Licence 4D View	Entier long	808465207
Licence 4D Write	Entier long	808464697
Licence SOAP 4D Client	Entier long	808465465
Licence Web 4D Client	Entier long	808465209

Passez dans le paramètre *groupe* le nom du groupe dont les utilisateurs seront autorisés à utiliser le plug-in.

Note : L'autorisation d'accès à un plug-in n'est accordée qu'à un seul groupe. Si un autre groupe disposait déjà de l'autorisation d'accès, il perd ce privilège à l'issue de l'exécution de la commande.

Ecrire proprietes groupe

Ecrire proprietes groupe (réfGroupe ; nom ; propriétaire {; membres}) -> Résultat

Paramètre	Type		Description
réfGroupe	Entier long	➡	Numéro de référence unique du groupe activé ou -1 pour ajouter un groupe de Super_Utilisateur -2 pour ajouter un groupe d'Administrateur
nom	Chaîne	➡	Nouveau nom de groupe
propriétaire	Entier long	➡	Numéro de référence unique de l'utilisateur ou le propriétaire du nouveau groupe
membres	Tableau entier long	➡	Nouveaux membres du groupe
Résultat	Entier long	↩	Numéro de référence unique du nouveau groupe

Description

Ecrire proprietes groupe vous permet de modifier et de mettre à jour les propriétés d'un groupe existant dont vous passez le numéro de référence unique dans *réfGroupe*, ou d'ajouter un nouveau groupe affilié au Super_Utilisateur ou à l'Administrateur.

Si vous modifiez les propriétés d'un groupe existant, vous devez passer son numéro de référence tel que retourné dans la commande **LIRE LISTE GROUPE**. Les numéros de référence de groupe sont les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15001 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Si vous voulez ajouter un nouveau groupe affilié au Super_Utilisateur, passez -1 dans *réfGroupe*. Si vous voulez ajouter un nouveau groupe affilié à l'Administrateur, passez -2 dans *réfGroupe*.

Si le groupe a bien été créé, **Ecrire proprietes groupe** retourne son numéro de référence unique.

Si vous ne passez pas -1, -2 ou un numéro de référence de groupe valide, **Ecrire proprietes groupe** ne fait rien et retourne 0.

Avant d'appeler cette routine, vous passez le nouveau nom du groupe et le numéro du propriétaire du groupe dans les paramètres *nom* et *propriétaire*.Si vous ne voulez pas modifier toutes les propriétés du groupe (à part ses membres, voir ci-dessous), passez les valeurs retournées par **LIRE PROPRIETES GROUPE** dans les paramètres que vous voulez laisser inchangés.

Si vous ne passez pas le paramètre optionnel *membres*, la liste courante des membres du groupe reste inchangée. Si vous le faites lors d'une création d'un groupe, le groupe n'aura pas de membres.

Note : Le propriétaire d'un groupe n'est pas automatiquement défini comme membre du groupe qu'il possède. C'est à vous de l'y inclure explicitement, à l'aide du paramètre *membres*.

Si vous passez le paramètre optionnel *membres*, vous modifiez toute la liste des membres pour ce groupe. Avant d'appeler cette routine, vous devez remplir le tableau *membres* avec les numéros de référence uniques des utilisateurs et/ou des groupes devant appartenir au groupe. Ces numéros peuvent être les suivants :

Numéro de référence du membre	Description membre
1	Super_Utilisateur
2	Administrateur
1 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15011 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Si vous voulez enlever tous les membres d'un groupe, passez un tableau vide dans le paramètre *membres*.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande **Ecrire proprietes groupe** ou si le système de mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par **APPELER SUR ERREUR**.

Ecrire proprietes utilisateur

Ecrire proprietes utilisateur (réfUtilisateur ; nom ; démarrage ; motDePasse ; nbUtilisations ; dernièreUtilisation {; adhésions {; groupePropriétaire}}) -> Résultat

Paramètre	Type		Description
réfUtilisateur	Entier long	⇒	Numéro de référence unique du compte de l'utilisateur ou -1 pour l'ajout d'un utilisateur affilié au Super_Utilisateur ou -2 pour l'ajout d'un utilisateur affilié à l'Administrateur
nom	Chaîne	⇒	Nouveau nom de l'utilisateur
démarrage	Chaîne	⇒	Nom de la nouvelle méthode de démarrage
motDePasse	Chaîne	⇒	Nouveau mot de passe (non crypté) ou * pour ne pas modifier le mot de passe
nbUtilisations	Entier long	⇒	Nouveau nombre d'utilisations de la base
dernièreUtilisation	Date	⇒	Nouvelle date de dernière utilisation de la base
adhésions	Tableau entier long	⇒	Numéros de référence des groupes auxquels l'utilisateur appartient
groupePropriétaire	Entier long	⇒	Numéro de référence du groupe propriétaire de l'utilisateur
Résultat	Entier long	⇐	Numéro de référence unique du nouvel utilisateur

Description

Ecrire proprietes utilisateur vous permet de modifier et de mettre à jour les propriétés d'un compte actif d'utilisateur existant dont le numéro de référence est passé dans le paramètre *réfUtilisateur*, ou d'ajouter un nouvel utilisateur affilié soit au Super_Utilisateur soit à l'Administrateur.

Si vous modifiez les propriétés d'un utilisateur existant, vous devez passer le numéro de référence qui vous est renvoyé par la commande **LIRE LISTE UTILISATEURS**.

Si le compte d'utilisateur n'existe pas ou a été supprimé, **Ecrire proprietes utilisateur** retourne 0 et l'erreur -9979 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par **APPELER SUR ERREUR**. Sinon, vous pouvez appeler la fonction **Utilisateur supprime** pour tester le compte de l'utilisateur avant d'appeler **Ecrire proprietes utilisateur**.

Les numéros de référence pour les utilisateurs peuvent être les suivants :

Numéro de référence de l'utilisateur	Description utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15000	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).

Si vous voulez ajouter un nouvel utilisateur affilié au Super_Utilisateur, il faut passer -1 à *réfUtilisateur*. Si vous voulez ajouter un nouvel utilisateur affilié à l'Administrateur, il faut passer -2 à *réfUtilisateur*.

Si l'utilisateur a bien été créé ou modifié, **Ecrire proprietes utilisateur** retourne son numéro de référence unique d'utilisateur.
Si vous ne passez pas un numéro de référence d'utilisateur valide, **Ecrire proprietes utilisateur** ne fait rien et retourne 0.

Lorsque vous appelez cette commande, vous passez le nouveau nom, la nouvelle méthode de démarrage, le nouveau mot de passe, le nouveau nombre d'utilisations et la nouvelle date de dernière utilisation pour l'utilisateur dans les paramètres *nom*, *démarrage*, *motDePasse*, *nbUtilisation* et *dernièreUtilisation*. Vous passez un mot de passe non crypté dans le paramètre *motDePasse*. 4D cryptera ce mot de passe avant de le sauvegarder dans le compte de l'utilisateur.
Si le nouveau nom d'utilisateur passé dans *nom* n'est pas unique (un utilisateur de même nom existe déjà), la commande ne fait rien et l'erreur -9979 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par **APPELER SUR ERREUR**.

Si vous ne voulez pas modifier toutes les propriétés de l'utilisateur (à part son groupe, voir ci-dessous), appelez au préalable **LIRE PROPRIETES UTILISATEUR** et passez les valeurs retournées dans celles que vous ne voulez pas modifier. Si vous ne voulez pas modifier le mot de passe de l'utilisateur, passez * dans le paramètre *motDePasse*. Cela vous permet de changer les autres propriétés du compte de l'utilisateur, sans changer le mot de passe de ce compte.

Si vous ne passez pas le paramètre optionnel *adhésions*, les adhésions de l'utilisateur restent inchangées. Si vous ne passez pas ce paramètre en cas d'ajout d'un utilisateur, il ne fera partie d'aucun groupe.

Si vous passez le paramètre optionnel *adhésions*, vous modifiez toutes les adhésions pour l'utilisateur. Avant d'appeler cette commande, vous devez remplir le tableau *adhésions* avec les numéros de référence uniques des groupes dont l'utilisateur devra faire partie.

Si vous passez le paramètre facultatif *groupePropriétaire*, vous indiquez le numéro de référence du groupe “propriétaire” de l'utilisateur, c'est-à-dire le groupe propriétaire par défaut des objets créés par cet utilisateur.

Les numéros de référence pour les groupes peuvent être les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15001 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Si vous voulez annuler les adhésions d'un utilisateur, passez un tableau vide dans le paramètre *adhésion*.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler **Ecrire proprietes utilisateur** ou si le système de mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par **APPELER SUR ERREUR**.

Licence disponible

Licence disponible {(licence)} -> Résultat

Paramètre	Type		Description
licence	Entier long	⇒	Plug-in duquel tester la validité de la licence
Résultat	Booléen	⇐	Vrai si le plug-in est disponible, sinon Faux

Description

La commande **Licence disponible** permet de connaître la disponibilité d'un plug-in. Elle est utile, par exemple, pour afficher ou masquer des fonctions nécessitant la présence d'un plug-in.

La commande **Licence disponible** peut être utilisée de trois manières différentes :

- Le paramètre *licence* est omis : dans ce cas, la commande retourne **Faux** si l'application 4D est en mode démonstration.
- Vous passez dans le paramètre *licence* une des constantes suivantes du thème "**Licence disponible**" :

Constante	Type	Valeur
Licence 4D for OCI	Entier long	808465208
Licence 4D Mobile	Entier long	808464439
Licence 4D ODBC Pro	Entier long	808464946
Licence 4D SOAP	Entier long	808465464
Licence 4D View	Entier long	808465207
Licence 4D Web	Entier long	808464945
Licence 4D Write	Entier long	808464697
Licence SOAP 4D Client	Entier long	808465465
Licence Test 4D Mobile	Entier long	808465719
Licence Web 4D Client	Entier long	808465209

  Dans ce cas, la commande retourne **Vrai** si le plug-in correspondant dispose d'une licence d'utilisation. La commande tient compte des éventuelles attributions de licences effectuées en mode Développement ou via la commande **ECRIRE ACCES PLUGIN**.
  Licence disponible retourne **Faux** si le plug-in fonctionne en mode démonstration.

- Vous passez directement dans le paramètre *licence* le numéro d'ID de la ressource "4BNX" du plug-in. Le fonctionnement de la commande est dans ce cas identique à celui décrit ci-dessus.

Lire acces plugin

Lire acces plugin (plugIn) -> Résultat

Paramètre	Type		Description
plugIn	Entier long	⇒	Numéro du plug-in
Résultat	Chaîne	⇐	Nom du groupe associé au plug-in

Description

La commande **Lire acces plugin** retourne le nom du groupe d'utilisateurs autorisé à utiliser le plug-in dont le numéro a été passé dans le paramètre *plugIn*. Si aucun groupe n'est associé au plug-in, la commande retourne une chaîne vide ("").

Passez dans le paramètre *plugIn* le numéro du plug-in duquel vous souhaitez connaître le groupe d'utilisateurs associé. Les licences de plug-ins incluent les licences Web et SOAP de 4D Client. Vous pouvez passer une des constantes suivantes du thème **Licence disponible** :

Constante	Type	Valeur
Licence 4D for OCI	Entier long	808465208
Licence 4D ODBC Pro	Entier long	808464946
Licence 4D View	Entier long	808465207
Licence 4D Write	Entier long	808464697
Licence SOAP 4D Client	Entier long	808465465
Licence Web 4D Client	Entier long	808465209

LIRE LISTE GROUPE

LIRE LISTE GROUPE (nomsGroupes ; numérosGroupes)

Paramètre	Type		Description
nomsGroupes	Tableau chaîne	⇐	Noms des groupes tels qu'ils apparaissent dans l'éditeur de Mots de passe
numérosGroupes	Tableau entier long	⇐	Numéros de référence uniques pour chaque groupe

Description

LIRE LISTE GROUPE remplit les tableaux *nomsGroupes* et *numérosGroupes* avec les noms et les numéros de référence uniques des groupes tels qu'ils apparaissent dans l'éditeur de Mots de passe.

Le tableau *numérosGroupes*, synchronisé avec le tableau *nomsGroupes*, est rempli avec les numéros de référence uniques des groupes. Ces numéros sont les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15011 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande **LIRE LISTE GROUPE** ou si le système de Mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par **APPELER SUR ERREUR**.

LIRE LISTE UTILISATEURS

LIRE LISTE UTILISATEURS (nomsUtil ; réfUtil)

Paramètre	Type		Description
nomsUtil	Tableau chaîne	⇐	Noms des utilisateurs tels qu'ils apparaissent dans l'éditeur de Mots de passe
réfUtil	Tableau entier long	⇐	Numéros de référence uniques pour chaque utilisateur

Description

La commande **LIRE LISTE UTILISATEURS** remplit les tableaux *nomsUtil* et *réfsUtil* avec les noms et les numéros de référence uniques des utilisateurs tels qu'ils apparaissent dans la fenêtre des Mots de passe de 4D.

Le tableau *nomsUtil* est rempli avec les noms des utilisateurs, y compris ceux dont le compte est supprimé (les utilisateurs dont le nom apparaît en vert dans la fenêtre des mots de passe).

Note : Utilisez la commande **Utilisateur supprime** pour savoir si un compte utilisateur est supprimé.

Le tableau *réfsUtil*, synchronisé avec *nomsUtil*, est rempli avec les numéros de référence uniques des utilisateurs. Ces numéros peuvent avoir les valeurs suivantes :

Numéro de référence de l'utilisateur	Description utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande **LIRE LISTE UTILISATEURS** ou si le système des Mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par **APPELER SUR ERREUR**.

LIRE PROPRIETES GROUPE

LIRE PROPRIETES GROUPE (réfGroupe ; nom ; propriétaire {; membres})

Paramètre	Type		Description
réfGroupe	Entier long	⇒	Numéro de référence du groupe
nom	Chaîne	⇐	Nom du groupe
propriétaire	Entier long	⇐	Numéro de référence du propriétaire du groupe
membres	Tableau entier long	⇐	Membres du groupe

Description

LIRE PROPRIETES GROUPE retourne les propriétés du groupe dont le numéro de référence est passé dans *réfGroupe*. Vous passez le numéro de référence du groupe retourné par la commande **LIRE LISTE GROUPE**. Les numéros de référence des groupes sont les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15011 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Si vous ne passez pas un numéro de référence valide, **LIRE PROPRIETES GROUPE** renvoie des paramètres vides.

Après l'appel de la commande, vous récupérez le nom et le numéro du propriétaire du groupe dans les paramètres *nom* et *propriétaire*.

Si vous passez le paramètre optionnel *membres*, ce tableau contiendra les numéros de référence uniques des utilisateurs qui appartiennent au groupe. Les numéros de référence des membres de groupe sont les suivants :

Numéro de référence du membre	Description membre
1	Super_Utilisateur
2	Administrateur
1 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15011 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande **LIRE PROPRIETES GROUPE** ou si le système de Mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par **APPELER SUR ERREUR**.

LIRE PROPRIETES UTILISATEUR

LIRE PROPRIETES UTILISATEUR (réfUtilisateur ; nom ; démarrage ; motDePasse ; nbUtilisations ; dernièreUtilisation {; adhésions {; groupePropriétaire}})

Paramètre	Type		Description
réfUtilisateur	Entier long	➡	Numéro de référence unique de l'utilisateur
nom	Chaîne	⬅	Nom de l'utilisateur
démarrage	Chaîne	⬅	Nom de la méthode de démarrage
motDePasse	Chaîne	⬅	*** obsolète (chaîne vide) ***
nbUtilisations	Entier long	⬅	Nombre d'utilisations de la base
dernièreUtilisation	Date	⬅	Date de la dernière utilisation de la base
adhésions	Tableau entier long	⬅	Numéros de référence des groupes auxquels l'utilisateur appartient
groupePropriétaire	Entier long	⬅	Numéro de référence du groupe propriétaire de l'utilisateur

Description

LIRE PROPRIETES UTILISATEUR retourne les informations concernant l'utilisateur dont le numéro de référence est passé dans le paramètre *réfUtilisateur*. Vous devez passer le numéro de référence retourné par la commande **LIRE LISTE UTILISATEURS**.

Si le compte d'utilisateur n'existe pas ou a été supprimé, l'erreur -9979 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par **APPELER SUR ERREUR**. Sinon, vous pouvez appeler la fonction **Utilisateur supprime** pour tester le compte de l'utilisateur avant d'appeler **LIRE PROPRIETES UTILISATEUR**.

Les numéros de référence pour les utilisateurs peuvent être les suivants :

Numéro de référence de l'utilisateur	Description utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).

Après l'appel, pour chaque utilisateur dont la référence est *réfUtilisateur*, vous récupérez aussi le nom, la méthode de démarrage, le nombre d'utilisations et la date de la dernière utilisation de la base dans les paramètres *nom*, *démarrage*, *nbUtilisation* et *dernièreUtilisation*.

Note : Le paramètre *motDePasse* est obsolète (il retourne toujours une chaîne vide). Si vous souhaitez contrôler le mot de passe d'un utilisateur, utilisez la fonction **Valider mot de passe**.

Si vous passez le paramètre facultatif *adhésion*, vous récupérez le numéro de référence unique du groupe auquel l'utilisateur appartient.

Si vous passez le paramètre facultatif *groupePropriétaire*, vous récupérez le numéro de référence du groupe “propriétaire” de l’utilisateur, c’est-à-dire le groupe propriétaire par défaut des objets créés par cet utilisateur.

Les numéros de référence pour les groupes peuvent être les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15001 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande **LIRE PROPRIETES UTILISATEUR** ou si le système de Mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par **APPELER SUR ERREUR**.

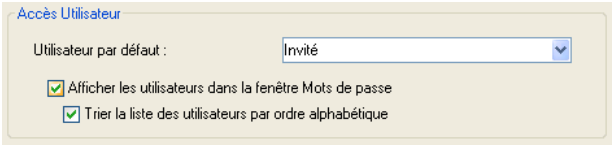
Lire utilisateur par defaut

Lire utilisateur par defaut -> Résultat

Paramètre	Type	Description
Résultat	Entier long	Numéro de référence unique de l'utilisateur

Description

La commande **Lire utilisateur par defaut** retourne le numéro de référence unique de l'utilisateur désigné comme "Utilisateur par défaut" dans la boîte de dialogue des Préférences de la base :



Les numéros de référence pour les utilisateurs peuvent être les suivants :

Numéro de référence de l'utilisateur	Description utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).

Si aucun utilisateur par défaut n'est défini, la commande retourne 0.

SUPPRIMER UTILISATEUR

SUPPRIMER UTILISATEUR (réfUtilisateur)

Paramètre	Type		Description
réfUtilisateur	Entier long	⇨	Numéro d'identification de l'utilisateur à supprimer

Description

La commande **SUPPRIMER UTILISATEUR** supprime l'utilisateur dont le numéro est passé dans *réfUtilisateur*. Vous devez passer un numéro valide d'utilisateur, retourné par la commande **LIRE LISTE UTILISATEURS**.

Si le compte de l'utilisateur n'existe pas ou a déjà été supprimé, une erreur -9979 est generée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

Seuls le Super_Utilisateur et l'Administrateur peuvent supprimer des utilisateurs. Il n'est pas possible à l'Administrateur de supprimer un utilisateur créé par le Super_Utilisateur.

Les utilisateurs supprimés n'apparaissent plus dans l'éditeur d'utilisateurs affiché lorsque vous appelez **CHANGER PRIVILEGES** ni en mode Développement. A noter que les numéros des utilisateurs supprimés peuvent être réattribués lors de la création de nouveaux comptes.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler **SUPPRIMER UTILISATEUR** ou si un autre process accède déjà au système de mots de passe, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par la commande **APPELER SUR ERREUR**.

Utilisateur courant

Utilisateur courant -> Résultat

Paramètre	Type	Description
Résultat	Chaîne	Nom de l'utilisateur courant

Description

Utilisateur courant retourne le "nom d'utilisateur" de l'utilisateur courant.

Exemple

Reportez-vous à l'exemple de la commande **Appartient au groupe**.

Utilisateur supprime

Utilisateur supprime (réfUtilisateur) -> Résultat

Paramètre	Type		Description
réfUtilisateur	Entier long	⇨	Numéro d'identification de l'utilisateur
Résultat	Booléen	⇦	Vrai = le compte de l'utilisateur est supprimé ou n'existe pas Faux = le compte de l'utilisateur est actif

Description

La commande **Utilisateur supprime** teste le compte de l'utilisateur dont le numéro d'identification unique est passé dans *réfUtilisateur*.
Si le compte n'existe pas ou a été supprimé, la fonction **Utilisateur supprime** retourne **Vrai**. Sinon, elle retourne **Faux**.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler **Utilisateur supprime** ou si un autre process accède déjà au système de mots de passe, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs utilisant la commande **APPELER SUR ERREUR**.

UTILISATEURS VERS BLOB

UTILISATEURS VERS BLOB (utilisateurs)

Paramètre	Type		Description
utilisateurs	BLOB	⇨	BLOB devant contenir les utilisateurs
		⇦	Comptes utilisateurs (crypté)

Description

La commande **UTILISATEURS VERS BLOB** stocke dans le BLOB *utilisateurs* la liste de tous les comptes d'utilisateurs et les groupes de la base créés par l'Administrateur.

Seuls l'Administrateur et le Super_Utilisateur de la base peuvent exécuter cette commande. Si un autre utilisateur tente de l'exécuter, la commande ne fait rien et une erreur de privilège (-9949) est générée.

Le BLOB généré est automatiquement encrypté et ne peut être lu que par la commande **BLOB VERS UTILISATEURS**. Vous pouvez stocker ce BLOB dans un fichier disque ou dans un champ.
Cette commande équivaut à l'enregistrement des groupes et utilisateurs depuis la fenêtre de gestion des groupes de la Boîte à outils, à la différence près qu'elle permet de stocker les comptes utilisateurs dans un champ BLOB et non uniquement dans un fichier.

Ce principe permet de conserver une sauvegarde des utilisateurs parmi les données de la base, et ainsi de mettre en place un mécanisme de sauvegarde et de chargement automatiques des utilisateurs en cas de mise à jour de la structure de la base (en effet, les informations relatives aux comptes utilisateurs sont stockées par 4D dans le fichier de structure de la base).

Valider mot de passe

Valider mot de passe (utilisateur ; motDePasse {; digest}) -> Résultat

Paramètre	Type		Description
utilisateur	Entier long, Chaîne	➡	N° de référence unique ou Nom de l'utilisateur
motDePasse	Chaîne	➡	Mot de passe non crypté
digest	Booléen	➡	Mot de passe digest = Vrai, Mot de passe en clair (défaut)= Faux
Résultat	Booléen	⮌	Vrai = mot de passe correct, Faux = mot de passe incorrect

Description

La commande **Valider mot de passe** retourne Vrai si la chaîne passée dans *motDePasse* est le mot de passe du compte utilisateur dont le n° de référence ou le nom est passé dans *utilisateur*.

Le paramètre optionnel *digest* vous permet d'indiquer si le paramètre *motDePasse* contient un mot de passe en clair ou un mot de passe sous forme hachée (mode digest) :

- si vous passez **Vrai**, vous indiquez que le paramètre *motDePasse* contient un mot de passe sous forme hachée (mode digest),
- si vous passez **Faux** ou omettez ce paramètre, vous indiquez que le paramètre *motDePasse* contient un mot de passe en clair.

Ce paramètre est particulièrement utile dans le contexte de l'utilisation des méthodes base d'authentification, notamment **Méthode base Sur authentification 4D Mobile**.

La commande est temporisée afin d'éviter des attaques par "force brute" (essais automatiques de multiples combinaisons de noms d'utilisateurs/mots de passe). Ainsi, au bout du quatrième appel à la commande, elle n'est exécutée qu'après 10 secondes d'attente. Cette temporisation est globale au poste de travail.

Exemple 1

L'exemple suivant vérifie que "Laurel" est le mot de passe de l'utilisateur "Hardy" :

```
LIRE LISTE UTILISATEURS(atNomUtil;alRefUtil)
$vlElem:=Chercher dans tableau(atNomUtil;"Hardy")
Si($vlElem>0)
   Si(Valider mot de passe(alRefUtil{$vlElem};"Laurel"))
      ALERTE("Oui !")
   Sinon
      ALERTE("Dommage !")
   Fin de si
Sinon
   ALERTE("Nom d'utilisateur inconnu")
Fin de si
```

Exemple 2

Dans la **Méthode base Sur authentification 4D Mobile**, vous souhaitez tester une requête de connexion (vous utilisez les utilisateurs 4D de la base). Il vous suffit d'écrire :

```
$0:=Valider mot de passe($1;$2;$3)
```

Variables

- ECRIRE VARIABLES
- EFFACER VARIABLE
- Indefinie
- LIRE VARIABLES

ECRIRE VARIABLES

ECRIRE VARIABLES (nomFichier ; variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Nom du document dans lequel sauvegarder la ou les variable(s)
variable	Variable	⇒	Variable(s) à sauvegarder

Description

La commande **ECRIRE VARIABLES** sauvegarde une ou plusieurs variable(s) dans un document disque dont le nom est passé dans le paramètre *document*.

Les variables ne doivent pas obligatoirement être du même type, mais doivent avoir le type Texte, numérique, Date, Heure, Booléen ou Image.

Si vous passez une chaîne vide ("") dans *document*, une boîte de dialogue standard d'enregistrement de fichiers apparaît, permettant à l'utilisateur de donner un nom au document à créer. Dans ce cas, la variable système *Document* récupère le nom du document, s'il a bien été créé.

Si les variables ont été correctement sauvegardées, la variable système OK prend la valeur 1. Sinon, OK prend la valeur 0.

Note : Lorsque vous écrivez des variables dans des documents à l'aide de la commande **ECRIRE VARIABLES**, 4D utilise un format de données qui lui est propre. Vous ne pouvez récupérer les variables qu'avec la commande **LIRE VARIABLES**. N'utilisez pas les commandes **RECEVOIR PAQUET** ou **RECEVOIR VARIABLE** pour lire un document créé par **ECRIRE VARIABLES**.

ATTENTION : La commande **ECRIRE VARIABLES** ne permet pas de sauvegarder les variables de type Tableau. Pour cela, vous devez utiliser les commandes du thème BLOB.

Exemple

L'exemple suivant enregistre trois variables dans un fichier nommé PrefsUti :

```
ECRIRE VARIABLES("PrefsUti";VSNom;VLCode;VGIconPict)
```

Variables et ensembles système

Si l'opération s'est correctement déroulée, la variable OK prend la valeur 1, sinon elle prend la valeur 0.

EFFACER VARIABLE

EFFACER VARIABLE (variable)

Paramètre	Type		Description
variable	Variable	➡	Nom de la variable à effacer

Description

EFFACER VARIABLE réinitialise *variable* à la valeur par défaut de son type (par exemple chaîne vide pour les types Alpha et Texte, 0 — zéro — pour les variables numériques, aucun élément pour un tableau etc.). La variable existe toujours en mémoire.

La variable passée dans *variable* peut être une variable locale, process ou interprocess.

Note : Il n'est pas nécessaire d'effacer les variables process à la fin de l'exécution d'un process, 4D s'en charge automatiquement. De même, chaque variable locale est automatiquement effacée à la fin de l'exécution de la méthode dans laquelle elle a été créée.

Exemple

Dans un formulaire, vous utilisez une liste déroulante appelée *asMalListeD* n'ayant qu'un rôle d'interface utilisateur. Autrement dit, vous exploitez ce tableau lors de la saisie de données, mais une fois que le formulaire est refermé, vous n'en avez plus besoin. Par conséquent, lors de l'événement Sur libération, vous effacez simplement le tableau :

```
 // Méthode objet liste déroulante asMalListeD
Au cas ou
   :(Evenement formulaire=Sur chargement)
 // Initialiser le tableau comme vous le souhaitez...
      TABLEAU ALPHA(63;asMalListeD;...)
 // ...
   :(Evenement formulaire=Sur libération)
 // Vous n'avez plus besoin du tableau
      EFFACER VARIABLE(asMalListeD)
 // ...
Fin de cas
```

Indefinie

Indefinie (variable) -> Résultat

Paramètre	Type		Description
variable	Variable	⇒	Variable à tester
Résultat	Booléen	⇐	Vrai = Variable actuellement indéfinie Faux = Variable actuellement définie

Description

Indefinie retourne **Vrai** si *variable* n'a pas été définie, et **Faux** si *variable* a été définie. Une variable est définie si elle a été créée via une directive de compilation ou si une valeur lui a été assignée. Elle est indéfinie dans les autres cas.

Si la base de données a été compilée, la fonction **Indefinie** retourne **Faux** pour toutes les variables.

Exemple

Votre application gère un process lorsqu'une commande de menu d'un module particulier de la base est sélectionnée : si le process est déjà créé, vous le passez au premier plan ; s'il n'est pas créé, vous le démarrez. Pour cela, pour chaque module de votre application, vous gérez une variable interprocess *<>PID_...* initialisée dans la **Méthode base Sur ouverture**.

Au cours du développement de la base, vous ajoutez de nouveaux modules. Au lieu de devoir à chaque fois modifier la **Méthode base Sur ouverture** (pour ajouter l'initialisation de la variable *<>PID_...* correspondante) puis quitter et réouvrir la base pour tout réinitialiser, vous utilisez la fonction **Indefinie** pour gérer "à la volée" l'ajout d'un nouveau module :

```
   // Méthode projet M_AJOUT_CLIENTS

Si(Indefinie(<>PID_AJOUT_CLIENTS)) // Prise en compte des étapes de développement intermédiaires
   C_ENTIER LONG(<>PID_AJOUT_CLIENTS)
   <>PID_AJOUT_CLIENTS:=0
Fin de si

Si(<>PID_AJOUT_CLIENTS=0)
   <>PID_AJOUT_CLIENTS:=Nouveau process("P_AJOUT_CLIENTS";64*1024;"P_AJOUT_CLIENTS")
Sinon
   MONTRER PROCESS(<>PID_AJOUT_CLIENTS)
   PASSER AU PREMIER PLAN(<>PID_AJOUT_CLIENTS)
Fin de si
 // Note: P_AJOUT_CLIENTS, la méthode de gestion des process,
 // fixe <>PID_ADD_CUSTOMERS à zéro lorsqu'elle est terminée.
```

LIRE VARIABLES

LIRE VARIABLES (nomFichier ; variable {; variable2 ; ... ; variableN})

Paramètre	Type		Description
nomFichier	Chaîne	⇨	Document contenant la ou les variable(s) à lire
variable	Variable	⇨	Nom de(s) variable(s) devant recevoir les valeurs

Description

La commande **LIRE VARIABLES** charge une ou plusieurs variable(s) depuis le document désigné par *document*. Ce document doit avoir été créé à l'aide de la commande **ECRIRE VARIABLES**.

Les variables *variable, variable2...variableN* sont soit créées, soit réécrites si elles existent déjà.

Si vous passez une chaîne vide ("") dans *document*, une boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de sélectionner le document à ouvrir. Dans ce cas, la variable système *Document* contiendra le nom du document choisi.

Dans le cadre de bases compilées, les variables utilisées doivent être du même type que celles chargées du disque.

ATTENTION : Cette commande ne traite pas les variables de type Tableau. Pour cela, vous devez utiliser les commandes du thème BLOB.

Exemple

L'exemple suivant charge trois variables d'un document nommé PrefsUti :

```
LIRE VARIABLES("PrefsUti";VSNom;VLCode;VGIconPict)
```

Variables et ensembles système

La variable système OK prend la valeur 1 si les variables ont été correctement chargées, sinon elle prend la valeur 0.

Web Services (Client)

- Commandes du thème Web Services (Client)
- WEB SERVICE APPELER
- WEB SERVICE AUTHENTIFIER
- WEB SERVICE FIXER OPTION
- WEB SERVICE FIXER PARAMETRE
- WEB SERVICE Lire infos
- WEB SERVICE LIRE RESULTAT

Commandes du thème Web Services (Client)

4D, à compter de la version 2003, prend en charge les “Web Services”, ce qui signifie que le programme vous permet de publier (partie serveur) et/ou d’utiliser (partie client) des Web Services depuis vos bases de données.
Un Web Service est un ensemble de fonctions publié sur un réseau. Ces fonctions peuvent être invoquées et utilisées par toute application compatible Web Services et connectée au réseau. Les Web Services peuvent effectuer tout type de tâche, comme le suivi d’acheminement de colis chez un transporteur, le commerce électronique, le suivi de valeurs boursières, etc.
Pour plus d’informations sur le concept et le fonctionnement des Web Services, reportez-vous au manuel Mode Développement.

La souscription aux Web Services dans 4D s’effectue simplement à l’aide de l’Assistant Web Services. Dans la plupart des cas, cet assistant sera suffisant pour vous permettre d’utiliser des Web Services. Toutefois, si vous souhaitez personnaliser certains mécanismes, vous devez utiliser les commandes SOAP client de 4D.

Cette section décrit les commandes utilisées pour la souscription par 4D à des Web Services externes (**partie client**). Pour plus d'informations sur les commandes utilisées lors de la publication des Web Services (partie serveur), reportez-vous aux **Commandes du thème Web Services (Serveur)**.

Note : Par convention, les libellés “SOAP” et “Web Service” ont été utilisés respectivement dans les noms des commandes (et des constantes) côté serveur et côté client. Ces deux notions désignent la même technologie.

WEB SERVICE APPELER

WEB SERVICE APPELER (urlAccès ; soapAction ; nomMéthode ; nameSpace {; typeComposé {; *}})

Paramètre	Type		Description
urlAccès	Chaîne	⇒	URL d'accès au Web Service
soapAction	Chaîne	⇒	Contenu du champ SOAPAction
nomMéthode	Chaîne	⇒	Nom de la méthode
nameSpace	Chaîne	⇒	Espace de nommage
typeComposé	Entier long	⇒	Configuration de types composés (types simples si omis)
*	Opérateur	⇒	Ne pas fermer la connexion

Description

La commande **WEB SERVICE APPELER** permet d'invoquer un Web Service en envoyant une requête HTTP. Cette requête contient le message SOAP préalablement construit à l'aide de la commande **WEB SERVICE FIXER PARAMETRE**.

Tout appel ultérieur à la commande **WEB SERVICE FIXER PARAMETRE** provoquera la construction d'une nouvelle requête. L'exécution d'une commande **WEB SERVICE APPELER** efface également tout éventuel résultat de Web Service précédemment appelé et le remplace par le(s) nouveau(x) résultats.

Passez dans *urlAccès* l'URL complet permettant d'accéder au Web Service (ne confondez pas cet URL avec celui du fichier WSDL, décrivant le Web Service).

- **Accès en mode sécurisé (SSL)** : Si vous souhaitez utiliser le Web Service en mode sécurisé (SSL), passez simplement https:// en tête de l'URL au lieu de http://. Ce paramétrage active automatiquement la connexion en mode sécurisé.
  A noter que cette commande peut utiliser un certificat serveur (cf. commande **HTTP FIXER DOSSIER CERTIFICATS**). Si le certificat n'est pas valide (expiré ou révoqué), la variable système OK prend la valeur 0 et l'erreur 901 "Certificat serveur invalide" est retournée. Vous pouvez intercepter cette erreur à l'aide d'une méthode d'appel sur erreur installée par la commande **APPELER SUR ERREUR**.

Passez dans le paramètre *soapAction* le contenu du champ SOAPAction de la requête. Ce champ contient généralement la valeur "**NomService#NomMéthode**".

Passez dans le paramètre *nomMéthode* le nom de la méthode distante (appartenant au Web Service) que vous souhaitez exécuter.

Passez dans le paramètre *nameSpace* l'espace de nommage XML utilisé pour la requête SOAP. Pour plus d'informations sur l'espace de nommage XML, reportez-vous au manuel *Mode Développement* de 4D.

Le paramètre optionnel *typeComposé* permet d'indiquer la configuration des paramètres Web Service envoyés ou reçus (définis à l'aide des commandes **WEB SERVICE FIXER PARAMETRE** et **WEB SERVICE LIRE RESULTAT**). La valeur du paramètre *typeComposé* dépend du mode de publication du Web Service (DOC ou RPC, cf. manuel *Mode Développement*) et de ses paramètres.
Vous devez passer dans *typeComposé* l'une des constantes suivantes, placées dans le thème **Web Services (Client)** :

Constante	Type	Valeur
Web Service dynamique	Entier long	0
Web Service entrée manuel	Entier long	1
Web Service manuel	Entier long	3
Web Service sortie manuel	Entier long	2

Chaque constante correspond à une "configuration" de Web Services. Une configuration représente une combinaison entre le mode de publication (RPC/DOC) et les types de paramètres entrée/sortie simples ou composés (aussi appelés "complexes") mis en oeuvre.

Note : La caractéristique "entrée" ou "sortie" des paramètres s'évalue du point de vue de la méthode proxy/du Web Service :
• un paramètre "entrée" est une valeur passée à la méthode proxy et donc au Web Service,
• un paramètre "sortie" est retourné par le Web Service et donc par la méthode proxy (généralement via $0).

Le tableau suivant fournit les configurations possibles et les constantes correspondantes :

	Paramètres entrée	
Paramètres sortie	**Simples**	**Composés**
Simples	RPC / Web Service dynamique	RPC / Web Service manuel entrée
Composés	RPC / Web Service manuel sortie	RPC ou DOC / Web Service manuel

Les cinq configurations décrites ci-dessous peuvent donc être mises en oeuvre. Dans tous les cas, 4D se charge de formater la requête SOAP à envoyer au Web Service ainsi que son enveloppe. Il vous appartient de formater le contenu de cette requête suivant la configuration utilisée.

Note : Bien qu'étant des types XML composés, les tableaux de données sont gérés par 4D comme des types simples.

Mode RPC, entrée et sortie simples

Cette configuration est la plus simple à utiliser. Dans ce cas, le paramètre *typeComposé* contient la constante Web Service dynamique ou est omis.
Les paramètres envoyés et les réponses reçues peuvent être manipulés directement, sans traitement préalable.
Reportez-vous à l'exemple de la commande **WEB SERVICE LIRE RESULTAT**.

Mode RPC, entrée composée et sortie simple

Dans ce cas, le paramètre *typeComposé* contient la constante Web Service entrée manuel. Avec cette configuration, vous devez passer "manuellement" au Web Service chaque élément xml source sous la forme d'un BLOB, à l'aide de la commande **WEB SERVICE FIXER PARAMETRE**.
Il vous appartient de formater le BLOB initial sous forme d'élément xml valide. Ce BLOB doit contenir comme premier élément le premier élément "fils" supposé de l'élément <Body> de la requête finale.

- Exemple

```
C_BLOB($1)
C_BOOLEEN($0)
```

```
WEB SERVICE FIXER PARAMETRE("MonBlobXML";$1)
WEB SERVICE APPELER("http://my.domain.com/mon_service";"MonSoapAction";"LaMethode";"http://my.namespace.com/";Web Service entrée manuel)
WEB SERVICE LIRE RESULTAT($0;"MaVarSortie";*)
```

Mode RPC, entrée simple et sortie composée

Dans ce cas, le paramètre *typeComposé* contient la constante Web Service sortie manuel. Chaque paramètre de sortie sera retourné par le Web Service sous forme d'élément xml stocké dans un BLOB. Vous récupérez ce paramètre à l'aide de la commande **WEB SERVICE LIRE RESULTAT**. Vous pourrez ensuite analyser le contenu du BLOB reçu à l'aide des commandes XML de 4D.

- Exemple

```
C_BLOB($0)
C_BOOLEEN($1)

WEB SERVICE FIXER PARAMETRE("MaVarEntree";$1)
WEB SERVICE APPELER("http://my.domain.com/mon_service";"MonSoapAction";"LaMethode";"http://my.namespace.com/";Web Service sortie manuel)
WEB SERVICE LIRE RESULTAT($0;"MonXMLSortie";*)
```

Mode RPC, entrée et sortie composées

Dans ce cas, le paramètre *typeComposé* contient la constante Web Service manuel. Chaque paramètre d'entrée et de sortie devra être stocké sous forme d'élément xml dans des BLOBs, comme décrit dans les deux configurations précédentes.

- Exemple

```
C_BLOB($0)
C_BLOB($1)

WEB SERVICE FIXER PARAMETRE("MonBlobXMLEntree";$1)
WEB SERVICE APPELER("http://my.domain.com/mon_service";"MonSoapAction";"LaMethode";"http://my.namespace.com/";Web Service manuel)
WEB SERVICE LIRE RESULTAT($0;"MonXMLSortie";*)
```

Mode DOC

Une méthode proxy d'appel d'un Web Service DOC est semblable à une méthode proxy d'appel d'un Web Service RPC utilisant des paramètres d'entrée et de sortie composés.
La seule différence entre ces deux configurations se situe au niveau du contenu xml des paramètres BLOB passés et reçus. Du point de vue de 4D, la construction et l'envoi de la requête SOAP sont identiques.

- Exemple

```
C_BLOB($0)
C_BLOB($1)

WEB SERVICE FIXER PARAMETRE("MonXMLEntree";$1)
WEB SERVICE APPELER("http://my.domain.com/mon_service";"MonSoapAction";"LaMethode";"http://my.namespace.com/";Web Service manuel)
WEB SERVICE LIRE RESULTAT($0;"MonXMLSortie";*)
```

Note : Dans le cas des Web Services DOC, la valeur des chaînes (ci-dessus "MonXMLEntree" et "MonXMLSortie") passées en paramètres n'a pas d'importance ; il est même possible de passer des chaînes vides (""). En effet, ces valeurs ne sont pas utilisées dans la requête SOAP contenant le document xml. Il est toutefois obligatoire de passer ces paramètres.

Pour utiliser un Web Service publié en mode DOC (ou en mode RPC avec types composés), il est conseillé de procéder de la manière suivante :

- Générer la méthode proxy à l'aide de l'Assistant Client Web Services.
  La méthode proxy sera appelée de la manière suivante : *$BlobXMLresult:=$proxy_MethodeDOC($BlobXMLparam)*
- Prendre connaissance du contenu des requêtes SOAP à envoyer au Web Service à l'aide d'un outil de test en ligne (par exemple *http://soapclient.com/soaptest.html*). Ce type d'outil permet, à partir du WSDL du Web Service, de générer des formulaires HTML de test.
- Copier le contenu xml généré à partir du premier fils de *<body>*.
- Ecrire la méthode permettant de placer les valeurs réelles des paramètres dans le code xml ; ce code doit ensuite être placé dans le BLOB *$BlobXMLparam*.
- Pour l'analyse de la réponse, vous pouvez également utiliser un outil de test en ligne, ou tirer parti du WSDL qui spécifie les éléments retournés.

Le paramètre * permet d'optimiser les appels. Lorsqu'il est passé, la commande ne referme pas la connexion utilisée par le process à l'issue de son exécution. Dans ce cas, l'appel suivant à **WEB SERVICE APPELER** réutilise cette même connexion si le paramètre * est passé, et ainsi de suite. Pour refermer la connexion, il suffit d'exécuter la commande **WEB SERVICE APPELER** sans le paramètre *. Ce mécanisme permet d'accélérer sensiblement les traitements en cas d'appels successifs de plusieurs Web Services sur le même serveur, notamment en configuration WAN (via Internet par exemple). A noter que ce mécanisme s'appuie sur le paramétrage "keep-alive" du serveur Web. Ce paramétrage définit généralement un nombre maximal de requêtes via une même connexion, et peut même les interdire. Si les requêtes **WEB SERVICE APPELER** enchaînées dans la même connexion atteignent ce nombre maximal ou si les connexions keep-alive ne sont pas autorisées, 4D créera une nouvelle connexion pour chaque requête.

Variables et ensembles système

Si la requête est correctement acheminée et que le Web Service l'a acceptée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 et une erreur est retournée.

WEB SERVICE AUTHENTIFIER

WEB SERVICE AUTHENTIFIER (nom ; motDePasse {; méthodeAuth} {; *})

Paramètre	Type		Description
nom	Chaîne	⇒	Nom de l'utilisateur
motDePasse	Chaîne	⇒	Mot de passe de l'utilisateur
méthodeAuth	Entier long	⇒	Méthode d'authentification : 0 ou omis=non définie, 1=BASIC, 2=DIGEST
*	Opérateur	⇒	Si passé : authentification par proxy

Description

La commande **WEB SERVICE AUTHENTIFIER** vous permet d'utiliser des Web Services nécessitant l'authentification de l'application cliente. Les méthodes BASIC et DIGEST sont prises en charge ainsi que la présence d'un proxy.

Note : Pour plus d'informations sur les méthodes d'authentification BASIC et DIGEST, reportez-vous à la section **Sécurité des connexions**.

Passez dans les paramètres *nom* et *motDePasse* les informations d'identification requises (nom d'utilisateur et mot de passe). Ces informations seront encodées et ajoutées à la requête HTTP envoyée au Web Service via la commande **WEB SERVICE APPELER**. Il est donc nécessaire d'appeler la commande **WEB SERVICE AUTHENTIFIER** avant la commande **WEB SERVICE APPELER**.

Le paramètre facultatif *méthodeAuth* permet d'indiquer la méthode d'authentification à utiliser pour le prochain appel de la commande **WEB SERVICE APPELER**. Vous pouvez passer l'une des valeurs suivantes :

- 2 = utiliser la méthode d'authentification DIGEST
- 1 = utiliser la méthode d'authentification BASIC
- 0 (ou paramètre omis) = utiliser la méthode appropriée. Dans ce cas, 4D envoie une requête supplémentaire afin de négocier la méthode d'authentification.

Si vous passez le paramètre *, vous indiquez que les informations d'authentification s'adressent à un proxy HTTP. Ce paramétrage doit être mis en oeuvre lorsqu'il existe un proxy nécessitant une authentification entre le client du Web Service et le Web Service lui-même. Si le Web Service est lui-même authentifié, une double authentification est requise (cf. exemple).

Par défaut, les informations d'authentification sont réinitialisées après chaque requête. Vous devez donc utiliser la commande **WEB SERVICE AUTHENTIFIER** avant chaque **WEB SERVICE APPELER**. Il est toutefois possible de conserver temporairement ces informations à l'aide d'une option de la commande **WEB SERVICE FIXER OPTION**. Dans ce cas, il n'est pas nécessaire d'exécuter la commande **WEB SERVICE AUTHENTIFIER** avant chaque **WEB SERVICE APPELER**.

En cas d'échec de l'authentification, le serveur SOAP retourne une erreur que vous pouvez identifier à l'aide de la commande **WEB SERVICE Lire infos**.

Exemple

Authentification auprès d'un Web Service situé derrière un proxy :

```
 //Authentification au Web Service en mode DIGEST
WEB SERVICE AUTHENTIFIER("SoapUser";"123";2)
 //Authentification au proxy en mode par défaut
WEB SERVICE AUTHENTIFIER("ProxyUser";"456";*)
WEB SERVICE APPELER(...)
```

WEB SERVICE FIXER OPTION

WEB SERVICE FIXER OPTION (option ; valeur)

Paramètre	Type		Description
option	Entier long	➡	Code de l'option à fixer
valeur	Entier long, Texte	➡	Valeur de l'option

Note préliminaire

Cette commande est destinée aux utilisateurs avancés des Web Services. Son emploi est facultatif.

Description

La commande **WEB SERVICE FIXER OPTION** permet de définir différentes options qui seront utilisées lors de la prochaine requête SOAP déclenchée par la commande **WEB SERVICE APPELER**.
Vous pouvez appeler cette commande autant de fois qu'il y a d'options à fixer.

Passez dans le paramètre *option* le numéro de l'option à définir et dans le paramètre *valeur* la nouvelle valeur de l'option. Vous pouvez utiliser pour ces deux paramètres une des constantes prédéfinies suivantes, situées dans le thème **Web Services (Client)** :

Constante	Type	Valeur	Comment
Web Service afficher dial auth	Entier long	4	*valeur* = 0 (ne pas afficher le dialogue) ou 1 (afficher le dialogue) Cette option gère l'affichage de boîte de dialogue d'authentification lors de l'exécution de la commande **WEB SERVICE APPELER**. Par défaut, cette commande ne provoque jamais l'affichage de la boîte de dialogue, vous devez en principe utiliser la commande **WEB SERVICE AUTHENTIFIER**. Toutefois, si vous souhaitez qu'une boîte de dialogue d'authentification apparaisse pour que l'utilisateur saisisse ses identifiants, vous devez utiliser cette option : passez 1 dans *valeur* pour afficher la boîte de dialogue, et 0 sinon. La boîte de dialogue n'apparaît que si le service Web requiert une authentification.
Web Service compression HTTP	Entier long	6	*valeur* = Web Service compression Cette option permet d'activer un mécanisme interne de compression des requêtes SOAP afin d'accélérer les échanges inter-applications 4D. Lorsque vous exécutez l'instruction **WEB SERVICE FIXER OPTION**(Web Service compression HTTP; Web Service compression) sur le client 4D du Web Service, les données de la prochaine requête SOAP envoyée par le client seront compressées en utilisant un mécanisme standard HTTP ("gzip" ou "deflate" en fonction du contenu de la requête) avant leur envoi au serveur SOAP 4D. Le serveur décompressera et analysera la requête puis répondra en utilisant automatiquement le même mécanisme. Seule la requête suivant l'appel de la commande **WEB SERVICE FIXER OPTION** est affectée. Vous devez donc appeler cette commande chaque fois que vous voulez utiliser la compression. Par défaut, 4D ne compresse pas les requêtes HTTP des Web Services. **Note :** Ce mécanisme ne peut pas être utilisé pour des requêtes adressées à un serveur SOAP 4D d'une version antérieure à la 11.3. Afin de vous permettre d'optimiser encore ce fonctionnement, des options supplémentaires configurent le seuil et le taux de compression des requêtes. Ces options sont accessibles via la commande **FIXER PARAMETRE BASE**.
Web Service effacer infos auth	Entier long	5	*valeur* = 0 (ne pas effacer les informations) ou 1 (les effacer) Cette option permet d'indiquer à 4D de mémoriser les informations d'authentification de l'utilisateur (nom d'utilisateur, mot de passe, méthode, etc.), dans le but de les réutiliser par la suite. Par défaut, ces informations sont effacées après chaque exécution de la commande **WEB SERVICE APPELER**. Passez 0 dans *valeur* pour les mémoriser et 1 pour les effacer. A noter que lorsque vous passez 0, les informations sont conservées pendant la session mais ne sont pas stockées.
Web Service header SOAP	Entier long	2	*valeur* = référence d'élément xml racine à insérer en tant que header (en-tête) de la requête SOAP. Cette option permet d'insérer un header dans la requête SOAP générée par la commande **WEB SERVICE APPELER**. Par défaut, les requêtes SOAP ne comportent pas d'en-tête spécifique. Cependant, certains Web Services requièrent la présence de cet en-tête, par exemple pour la gestion de paramètres d'identification.
Web Service timeout HTTP	Entier long	1	*valeur* = "timeout" de la partie cliente exprimé en secondes. Le timeout de la partie cliente est le délai d'attente du client Web Service en cas de non-réponse du serveur. A l'issue de ce délai, le client referme la session, la requête est perdue. Par défaut, ce délai est de 180 secondes. Il peut être modifié en raison de caractéristiques particulières (état du réseau, spécificités du Web Service, etc.).
Web Service version SOAP	Entier long	3	*valeur* = Web Service SOAP_1_1 ou Web Service SOAP_1_2 Cette option permet de préciser la version du protocole SOAP utilisée dans la requête. Passez dans valeur la constante Web Service SOAP_1_1 pour indiquer la version 1.1 et la constante Web Service SOAP_1_2 pour indiquer la version 1.2.

L'ordre d'appel des options n'a pas d'importance. Si une même *option* est définie plusieurs fois, seule la valeur du dernier appel est prise en compte.

Exemple 1

Insertion d'un en-tête personnalisé dans la requête SOAP :

```
  ` Création d'une référence XML
C_TEXTE(vRefRacine;vRefElement)
vRefRacine:=DOM Creer ref XML("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElement:=DOM Creer element XML(vRefRacine;vxPath)
  `Modification de l'en-tête SOAP avec la référence
WEB SERVICE FIXER OPTION(Web Service header SOAP;vRefElement)
```

Exemple 2

Utilisation de la version 1.2 du protocole SOAP :

```
WEB SERVICE FIXER OPTION(Web Service version SOAP;Web Service SOAP_1_2)
```

WEB SERVICE FIXER PARAMETRE

WEB SERVICE FIXER PARAMETRE (nom ; valeur {; typeSOAP})

Paramètre	Type		Description
nom	Chaîne	➡	Nom du paramètre à inclure dans la requête SOAP
valeur	Variable	➡	Variable 4D contenant la valeur du paramètre
typeSOAP	Chaîne	➡	Type SOAP du paramètre

Description

La commande **WEB SERVICE FIXER PARAMETRE** permet de définir un paramètre utilisé pour une requête SOAP cliente. Appelez autant de fois cette commande qu'il y a de paramètres dans la requête.

Passez dans *nom* le nom du paramètre tel qu'il doit apparaître dans la requête SOAP.

Passez dans *valeur* la variable 4D contenant la valeur du paramètre. Dans le cadre des méthodes proxy, cette variable est généralement *$1*, *$2*, *$3*, etc., correspondant à un paramètre 4D passé à la méthode proxy lors de son appel. Il est toutefois possible d'utiliser des variables intermédiaires.

Note : Chaque variable ou tableau 4D utilisé(e) doit être préalablement déclaré(e) à l'aide des commandes des thèmes **Compilateur** et **Tableaux**.

Par défaut, 4D détermine automatiquement le type SOAP le plus approprié pour le paramètre *nom* en fonction du contenu de *valeur*. L'indication du type est incluse dans la requête.
Toutefois, vous pouvez vouloir "forcer" la définition du type SOAP du paramètre. Dans ce cas, vous pouvez passer le paramètre optionnel *typeSOAP* ; vous devez utiliser une des chaînes de caractères suivantes (types de données primaires) :

typeSOAP	Description
string	Chaîne
int	Entier long
boolean	Booléen
float	Réel 32 bits
decimal	Réel avec décimale
double	Réel 64 bits
duration	Durée en années mois jours heures minutes secondes, par exemple : P1Y2M3DT10H30M
datetime	Date et heure au format ISO8601, par exemple 2003-05-31T13:20:00
time	Heure, par exemple 13:20:00
date	Date, par exemple 2003-05-31
gyearmonth	Année et mois (calendrier grégorien), par exemple 2003-05
gyear	Année (calendrier grégorien), par exemple 2003
gmonthday	Mois et jour (calendrier grégorien), par exemple --05-31
gday	Jour (calendrier grégorien), par exemple ---31
gmonth	Mois (calendrier grégorien), par exemple --10--
hexbinary	Valeur exprimée en hexadécimal
base64binary	BLOB
anyuri	Uniform Resource Identifier (URI), par exemple : http://www.societe.com/page
qname	Nom XML qualifié (espace de nommage et partie locale)
notation	Attribut Notation

Note : Pour plus d'informations sur les types de données XML, reportez-vous à l'URL *http://www.w3.org/TR/xmlschema-2/*

Exemple

Cet exemple définit deux paramètres :

```
C_TEXTE($1)
C_TEXTE($2)
WEB SERVICE FIXER PARAMETRE("ville";$1)
WEB SERVICE FIXER PARAMETRE("pays";$2)
```

WEB SERVICE Lire infos

WEB SERVICE Lire infos (typeInfo) -> Résultat

Paramètre	Type		Description
typeInfo	Entier long	⇒	Information à récupérer
Résultat	Chaîne	⇐	Information sur la dernière erreur SOAP

Description

La commande **WEB SERVICE Lire infos** retourne des informations relatives à l'erreur éventuellement générée lors de l'exécution de la dernière requête SOAP adressée à un Web Service. Cette commande doit généralement être appelée dans le cadre d'une méthode d'appel sur erreur installée par la commande **APPELER SUR ERREUR**.

Le paramètre *typeInfo* vous permet d'indiquer le type d'information que vous souhaitez obtenir. Vous devez passer une des constantes suivantes, placées dans le thème **Web Services (Client)** :

Constante	Type	Valeur	Comment
Web Service code erreur	Entier long	0	Code d'erreur principal (défini par 4D). Ce code est également retourné dans la variable système Error. Voici la liste des codes pouvant être retournés : 9910 : Erreur Web Service (voir aussi Web Service origine erreur) 9911 : Erreur de l'analyseur xml 9912 : Erreur HTTP (voir aussi Web Service code erreur HTTP) 9913 : Erreur réseau 9914 : Erreur interne
Web Service code statut HTTP	Entier long	2	Code de statut HTTP (à utiliser en cas d'erreur principale 9912).
Web Service message	Entier long	1	Message détaillé décrivant l'erreur. Le type de message diffère suivant le type d'erreur principale. - Si erreur principale = 9910 (Erreur Web Service) : la cause de l'erreur SOAP est retournée (ex : "méthode appelée inexistante"). - Si erreur principale = 9911 (Erreur de l'analyseur xml) : l'emplacement de l'erreur dans le document xml est retourné. - Si erreur principale = 9912 (Erreur HTTP) : - si l'erreur HTTP est située dans l'intervalle [300-400] (problèmes lié à l'emplacement du document demandé), le nouvel emplacement de l'URL demandé est retourné. - pour tout autre code d'erreur HTTP, le <body> est renvoyé. - Si erreur principale = 9913 (Erreur réseau) : la cause de l'erreur réseau est retournée (ex : "AdresseServeur : erreur DNS") - Si erreur principale = 9914 (Erreur interne) : la cause de l'erreur interne est retournée
Web Service origine erreur	Entier long	3	Cause de l'erreur (retournée par le protocole SOAP — à utiliser en cas d'erreur principale 9910). - Version Mismatch (les versions ne correspondent pas). - Must Understand (un paramètre défini comme obligatoire n'a pas pu être interprété par le serveur) - Sender Fault - Receiver Fault - Encoding Unknown (encodage inconnu)

Une chaîne vide est retournée lorsqu'aucune information n'est disponible, en particulier si la dernière requête SOAP n'a pas généré d'erreur.

WEB SERVICE LIRE RESULTAT

WEB SERVICE LIRE RESULTAT (valeurRetour {; nomRetour {; *}})

Paramètre	Type		Description
valeurRetour	Variable	⇦	Valeur retournée par le Web Service
nomRetour	Chaîne	⇨	Nom du paramètre à récupérer
*		⇨	Libérer la mémoire

Description

La commande **WEB SERVICE LIRE RESULTAT** permet de récupérer une valeur renvoyée par le Web Service à l'issue du traitement effectué.

Note : Cette commande doit être utilisée uniquement après la commande **WEB SERVICE APPELER**.

Le paramètre *valeurRetour* reçoit la valeur renvoyée par le Web Service. Passez dans ce paramètre une variable 4D. Cette variable est généralement *$0*, correspondant à la valeur retournée par la méthode proxy. Il est toutefois possible d'utiliser des variables intermédiaires (vous devez utiliser des variables process uniquement).

Note : Chaque variable ou tableau 4D utilisé(e) doit être préalablement déclaré(e) à l'aide des commandes des thèmes "Compilateur" et "Tableaux".

Le paramètre optionnel *nomRetour* permet de spécifier le nom du paramètre à récupérer. Toutefois, comme la plupart des Web Services retournent une seule valeur, ce paramètre n'est généralement pas nécessaire.

Le paramètre *, optionnel, indique au programme de libérer la mémoire consacrée au traitement de la requête. Vous devez passer ce paramètre après la récupération de la dernière valeur renvoyée par le Web Service.

Exemple

Imaginons un Web Service retournant l'heure courante dans n'importe quelle ville du monde. Les paramètres reçus par le Web Service sont le nom de la ville et le code du pays. Le Web Service retourne alors l'heure correspondante. La méthode proxy d'appel pourrait être de la forme suivante :

```
C_TEXTE($1)
C_TEXTE($2)
C_HEURE($0)

WEB SERVICE FIXER PARAMETRE("ville";$1)
WEB SERVICE FIXER PARAMETRE("code_pays";$2)

WEB SERVICE APPELER("http://www.villesdumonde.com/WS";"WSHeures#Heure_ville";"Heure_ville";
                    "http://www.villesdumonde.com/namespace/default")

Si(OK=1)
   WEB SERVICE LIRE RESULTAT($0;"retour";*)
Fin de si
```

Web Services (Serveur)

- Commandes du thème Web Services (Serveur)
- SOAP DECLARATION
- SOAP ENVOYER ERREUR
- SOAP Lire information
- SOAP Requete

Commandes du thème Web Services (Serveur)

La publication de Web Services dans 4D s'effectue simplement à l'aide d'options dans les propriétés des méthodes. Dans la plupart des cas, ce fonctionnement sera suffisant pour vous permettre de publier des Web Services. Toutefois, si vous souhaitez personnaliser certains mécanismes, utiliser des tableaux de données..., vous devez utiliser les commandes SOAP serveur de 4D.

Cette section décrit les commandes utilisées lors de la publication dans 4D de Web Services (**partie serveur**). Pour des informations générales sur les Web Services ou la description des commandes utilisées lors de la souscription aux Web Services (partie client), reportez-vous à la section **Commandes du thème Web Services (Client)**.

Note : Les libellés "SOAP" et "Web Service" ont été utilisés respectivement dans les noms des commandes (et des constantes) côté serveur et côté client par convention uniquement. Ces deux notions désignent la même technologie.

SOAP DECLARATION

SOAP DECLARATION (variable ; type ; entrée_sortie {; alias})

Paramètre	Type		Description
variable	Variable	➡	Variable référençant un argument SOAP entrant ou sortant
type	Entier long	➡	Type 4D vers lequel pointe l'argument
entrée_sortie	Entier long	➡	1 = Entrée SOAP, 2 = Sortie SOAP
alias	Chaîne	➡	Nom publié pour cet argument lors des échanges SOAP

Description

La commande **SOAP DECLARATION** permet de déclarer explicitement le type des paramètres utilisés dans une méthode 4D publiée comme Web Service.

Lorsqu'une méthode est publiée en tant que Web Service, les paramètres standard *$0*, *$1*... *$n* sont utilisés pour décrire les paramètres du Web Service auprès du monde extérieur (notamment dans le fichier WSDL). Le protocole SOAP exigeant que les paramètres soient explicitement nommés, 4D utilise par défaut les noms "FourD_arg0, FourD_arg1 ... FourD_argn".

Ce fonctionnement par défaut peut toutefois s'avérer problématique pour les raisons suivantes :

- Il n'est pas possible de déclarer *$0* ou *$1*, *$2*, etc. en tant que tableau. Il est donc nécessaire d'utiliser des pointeurs, mais dans ce cas il reste à connaître le type des valeurs pour la génération du fichier WSDL.
- Il peut être utile ou nécessaire de personnaliser les noms des paramètres (entrants et sortants).
- Vous pouvez vouloir utiliser des paramètres tels que des structures XML et des références DOM.
- Il peut également être nécessaire de retourner des valeurs d'une taille supérieure à 32 Ko (limite des arguments de type Texte dans les bases de données en mode non Unicode).
- Enfin, ce fonctionnement rend impossible le retour de plus d'une valeur par appel (dans *$0*).

La commande **SOAP DECLARATION** permet de s'affranchir de ces limites. Vous pouvez exécuter cette commande pour chaque paramètre entrant et sortant et lui assigner un nom et un type.

Note : Même si la commande **SOAP DECLARATION** est utilisée, il est nécessaire de déclarer les variables et tableaux 4D dans la méthode Compiler_Web à l'aide les commandes du thème "Compilateur".

Passez dans *variable* la variable 4D à référencer dans l'appel du Web Service.

Attention, vous pouvez référencer uniquement des variables process ou les arguments des méthodes 4D (*$0* à *$n*). Les variables locales et interprocess ne peuvent pas être utilisées.

Par défaut, seuls des arguments de type Texte pouvant être utilisés, les réponses du serveur SOAP sont en principe limitées à 32 Ko dans les bases de données en mode non Unicode. Il est toutefois possible de retourner des arguments SOAP d'une taille supérieure à 32 Ko, en utilisant des BLOBs. Pour cela, il suffit de déclarer les arguments en type BLOB avant d'appeler la commande **SOAP DECLARATION** (cf. exemple 4).

Note : Côté client, si vous souscrivez à ce type de Web Service avec 4D, l'assistant Web Services générera naturellement une variable de type Texte dans la méthode proxy. Pour pouvoir l'utiliser, il vous suffit de retyper cette variable de retour en BLOB dans la méthode proxy.

Passez dans *type* le type 4D correspondant. La plupart des types de variables et de tableaux 4D peuvent être employés. Vous pouvez utiliser les constantes ci-dessous, placées dans le thème **Types champs et variables**, ainsi que, pour les types XML, deux constantes du thème **Web Services (Serveur)** :

Constante	Type	Valeur
Est un BLOB	Entier long	30
Est un booléen	Entier long	6
Est un entier	Entier long	8
Est un entier long	Entier long	9
Est un numérique	Entier long	1
Est un tableau booléen	Entier long	22
Est un tableau chaîne	Entier long	21
Est un tableau date	Entier long	17
Est un tableau entier	Entier long	15
Est un tableau entierlong	Entier long	16
Est un tableau numérique	Entier long	14
Est un tableau texte	Entier long	18
Est un texte	Entier long	2
Est une date	Entier long	4
Est une heure	Entier long	11
Est une variable chaîne	Entier long	24

Constante	Type	Valeur	Comment
Est un XML	Entier long	36	
Est une référence DOM	Entier long	37	

Passez dans *entrée_sortie* une valeur indiquant si le paramètre traité est "entrant" (c'est-à-dire correspondant à une valeur reçue par la méthode) ou "sortant" (c'est-à-dire correspondant à une valeur retournée par la méthode). Vous pouvez utiliser les constantes prédéfinies suivantes, placées dans le thème "**Web Services (Serveur)**" :

Constante	Type	Valeur
SOAP entrée	Entier long	1
SOAP sortie	Entier long	2

Utilisation de types XML

Vous pouvez déclarer des variables de type "structure XML" et "référence DOM", aussi bien en entrée qu'en sortie, via les constantes Est un XML et Est une référence DOM. Lorsque des paramètres de ce type sont définis, aucun traitement ni encodage ne leur est appliqué, les données sont transmises telles

quelles (cf. exemple 5).

- Paramètres sortants :
  - Est un XML indique que le paramètre contient une structure XML,
  - Est une référence DOM indique que le paramètre contient la référence DOM d'une structure XML. Dans ce cas, l'insertion de la structure XML dans le message SOAP équivaut à l'exécution de la commande **DOM EXPORTER VERS VARIABLE**.

Note : Dans le cas de références DOM utilisées en paramètres sortants, il est recommandé d'utiliser des références globales, créées par exemple au démarrage et closes à la fermeture de l'application. En effet, une référence DOM créée au sein du Web Service lui-même ne peut pas être refermée avec **DOM FERMER XML** sinon le Web Service ne retourne plus rien. Les appels multiples au Web Service impliquent alors la création d'autant de références DOM non refermées, ce qui peut provoquer une saturation de la mémoire.

- Paramètres entrants :
  - Est un XML indique que le paramètre doit recevoir un argument XML envoyé par le client SOAP.
  - Est une référence DOM indique que le paramètre doit recevoir la référence DOM d'une structure XML correspondant à l'argument XML envoyé par le client SOAP.

- Modification de la WSDL : Les structures XML seront déclarées par 4D du type "anyType" (indéterminé) dans la WSDL. Si vous souhaitez typer précisément une structure XML, vous devez sauvegarder le fichier WSDL et ajouter manuellement le schéma de données souhaité dans la section <types> de la WSDL.

Méthode COMPILER_WEB

Les arguments SOAP entrants référencés à l'aide de variables 4D (et non via les arguments des méthodes 4D) doivent être préalablement déclarés dans la méthode projet COMPILER_WEB. En effet, l'utilisation de variables process dans les méthodes Web Services nécessite leur déclaration avant l'appel de la méthode. La méthode projet COMPILER_WEB est appelée, si elle existe, à chaque requête SOAP acceptée. Par défaut, la méthode COMPILER_WEB n'existe pas. Vous devez la créer explicitement.
A noter que la méthode COMPILER_WEB est également appelée par le serveur Web de 4D lors de la réception de requêtes Web "classiques" de type POST (cf. section **URLs et actions de formulaires**).

Passez dans *alias* le nom de l'argument tel qu'il doit apparaître dans la WSDL et dans les échanges SOAP. **Attention**, ce nom doit être unique dans l'appel RPC (paramètres entrants et sortants confondus), sinon seule la dernière déclaration sera prise en compte par 4D.

Note : Les noms des arguments ne doivent pas débuter par un chiffre ni contenir d'espaces. En outre, pour éviter tout risque d'incompatibilité, il est recommandé de ne pas utiliser de caractères étendus (tels que des caractères accentués).

Si le paramètre *alias* est omis, 4D utilisera par défaut le nom de la variable ou **FourD_argN** pour les arguments des méthodes 4D (*$0* à *$n*).

Note : La commande **SOAP DECLARATION** doit être incluse dans la méthode publiée comme Web Service. Il n'est pas possible de l'appeler d'une autre méthode.

Exemple 1

Cet exemple spécifie un nom de paramètre :

```
 //Dans la méthode COMPILER_WEB
C_ENTIER LONG($1)

 //Dans la méthode du service Web
SOAP DECLARATION($1;Est un entier long;SOAP entrée;"zipcode")
 //Lors de la génération du fichier WSDL et des appels SOAP, le libellé zipcode sera utilisé au lieu de fourD_arg1
```

Exemple 2

Cet exemple permet de récupérer un tableau de codes postaux sous forme d'entiers longs :

```
 //Dans la méthode COMPILER_WEB
TABLEAU ENTIER LONG(tab_codes;0)

 //Dans la méthode du service Web
SOAP DECLARATION(tab_codes;Est un tableau entierlong;SOAP entrée;"in_tab_codes")
```

Exemple 3

Cet exemple permet de référencer deux valeurs de retour sans spécifier de nom d'argument :

```
SOAP DECLARATION(ret1;Est un entier long;SOAP sortie)
SOAP DECLARATION(ret2;Est un entier long;SOAP sortie)
```

Exemple 4

Cet exemple permet au serveur SOAP de 4D de retourner un argument d'une taille supérieure à 32 Ko dans les bases de données en mode non Unicode :

```
C_BLOB($0)
SOAP DECLARATION($0;Est un texte;SOAP sortie)
```

Notez le type Est un texte (et non Est un BLOB). Cette astuce permet un formatage correct de l'argument.

Exemple 5

Cet exemple illustre l'effet des différents types de déclarations :

```
TOUT SELECTIONNER([Contact])

 //Construction d'une structure XML à partir de la sélection de Contacts et stockage du XML dans un BLOB
C_BLOB(ws_vx_xmlBlob)
getContactsXML(->ws_vx_xmlBlob)
 //Récupération de la structure XML dans une variable texte
C_TEXTE(ws_vt_xml)
ws_vt_xml:=BLOB vers texte(ws_vx_xmlBlob;UTF8 texte sans longueur)
 //Récupération d'une référence DOM vers la structure XML
C_TEXTE(ws_vt_refXML)
ws_vt_refXML:=DOM Analyser variable XML(ws_vt_xml)

 //Test des différentes déclarations
SOAP DECLARATION(ws_vx_xmlBlob;Est un BLOB;SOAP sortie;"contactListsX")
 //Le XML est converti en Base64 par 4D

SOAP DECLARATION(ws_vt_xml;Est un texte;SOAP sortie;"contactListsText")
 //Le XML est converti en texte par 4D (< > deviennent des entités)

SOAP DECLARATION(ws_vt_xml;Est un XML;SOAP sortie;"contactsXML")
 //Le XML est passé en texte XML

SOAP DECLARATION(ws_vx_xmlBlob;Est un XML;SOAP sortie;"contactsBlob")
 //Le XML est passé en BLOB XML

SOAP DECLARATION(ws_vt_refXML;Est une référence DOM;SOAP sortie;"contactByRef")
 //Le XML est passé en tant que référence
```

SOAP ENVOYER ERREUR

SOAP ENVOYER ERREUR (typeErreur ; description)

Paramètre	Type		Description
typeErreur	Entier long	⇒	1 = Erreur Client, 2 = Erreur Serveur
description	Chaîne	⇒	Description de l'erreur à envoyer au client SOAP

Description

La commande **SOAP ENVOYER ERREUR** permet de retourner une erreur à un client SOAP en indiquant l'origine de l'erreur : client ou serveur. Utiliser cette commande vous permet de signaler une erreur à un client sans devoir retourner de résultat.

Par exemple, une erreur côté client peut être détectée lorsque vous publiez un Web Service "Racine_carree" et qu'un client envoie une requête avec un nombre négatif ; vous pouvez utiliser cette commande afin d'indiquer au client qu'une valeur positive est requise.

Une erreur possible côté serveur peut être par exemple un manque de mémoire survenu lors de l'exécution de la méthode.

Passez dans *typeErreur* l'origine de l'erreur. Vous pouvez utiliser les constantes prédéfinies suivantes, placées dans le thème **Web Services (Serveur)** :

Constante	Type	Valeur
SOAP erreur client	Entier long	1
SOAP erreur serveur	Entier long	2

Passez dans *description* un descriptif de l'erreur. Si l'implémentation du client est conforme, l'erreur pourra être traitée.

Exemple

Pour reprendre l'exemple du Web Service "Racine_carree" fourni dans la description de la commande, l'instruction suivante peut être utilisée pour traiter les requêtes sur des nombres négatifs :

```
SOAP ENVOYER ERREUR(SOAP erreur client;"Valeurs positives requises")
```

SOAP Lire information

SOAP Lire information (numInfo) -> Résultat

Paramètre	Type		Description
numInfo	Entier long	⇒	Numéro du type d'information SOAP à lire
Résultat	Chaîne	⇐	Information SOAP

Description

La commande **SOAP Lire information** permet de récupérer sous forme de chaîne de caractères différents types d'informations concernant une requête SOAP.

Lorsque vous traitez une requête SOAP, il peut être utile d'obtenir des informations supplémentaires — en-dehors des valeurs des paramètres RPC — sur la requête. Par exemple, pour des raisons de sécurité, vous pouvez utiliser cette commande dans la **Méthode base Sur authentification Web** afin de connaître le nom de la méthode Web Service demandée.

Passez dans le paramètre *numInfo* le numéro du type d'information SOAP à connaître. Vous pouvez utiliser les constantes prédéfinies suivantes, placées dans le thème **Web Services (Serveur)** :

Constante	Type	Valeur	Comment
SOAP nom méthode	Entier long	1	Nom de la méthode offerte comme Web Service sur le point d'être exécutée
SOAP nom service	Entier long	2	Nom du Web Service auquel appartient la méthode

Note : Pour des raisons de sécurité également, il est possible de définir la taille maximale des requêtes Web Services adressées à 4D. Ce paramétrage est effectué à l'aide de la commande **FIXER PARAMETRE BASE** (thème "Définition structure").

SOAP Requete

SOAP Requete -> Résultat

Paramètre	Type		Description
Résultat	Booléen		Vrai si la requête est SOAP, Faux sinon

Description

La commande **SOAP Requete** retourne **Vrai** si le code en cours d'exécution fait partie d'une requête SOAP.

Cette commande peut être utilisée pour des raisons de sécurité dans la **Méthode base Sur authentification Web** afin de déterminer la nature des requêtes reçues.

XML

Présentation des commandes XML génériques

Ce thème regroupe les commandes XML génériques "utilitaires" de 4D. Il s'agit des commandes de gestion d'erreurs et d'options ainsi que des commandes (obsolètes depuis 4D v14 R4) spécialisées dans le XSL.

Pour des informations générales sur le XML (présentation, glossaire) ainsi que sur les différences entre les modes DOM et SAX, veuillez vous reporter à la section **Présentation des commandes XML DOM**.

Qu'est-ce que le SVG ?

SVG (Scalable Vector Graphics) est un format de fichier permettant de décrire en XML un graphique vectoriel (extension .svg). L'utilisation la plus courante du SVG est la publication de données statistiques ou cartographiques.
Ces fichiers peuvent être visualisés dans les navigateurs Web, soit nativement, soit via des plug-ins. 4D v11 comporte un moteur de rendu SVG vous permettant de visualiser les fichiers SVG dans les champs ou les variables image. La commande **SVG EXPORTER VERS IMAGE** vous permet de générer une image dans 4D à partir d'une description SVG. A noter également que la commande **GRAPHE** permet de tirer parti du moteur SVG intégré de 4D.
Pour plus d'informations sur ce format, reportez-vous à l'adresse http://www.w3.org/Graphics/SVG/.

A propos des commandes XSLT

A compter de 4D v14 R4, les commandes de transformation XSL sont déclarées obsolètes et ont été préfixées en conséquence :

Ancien nom	4D v14 R4 et suivantes
XSLT APPLIQUER TRANSFORMATION	**_o_XSLT APPLIQUER TRANSFORMATION**
XSLT LIRE ERREUR	**_o_XSLT LIRE ERREUR**
XSLT FIXER PARAMETRE	**_o_XSLT FIXER PARAMETRE**

Par compatibilité, les transformations XSL sont toujours prises en charge dans 4D, mais leur usage est désormais déconseillé. L'utilisation de la technologie XSLT ne sera plus possible dans les versions futures de 4D.

Note 4D Server 64-bit OS X : Le XSLT n'est pas inclus dans 4D Server 64-bit pour OS X. Par conséquent, l'exécution d'une de ces commandes depuis cette application génère l'erreur 33, "Méthode ou fonction non implémentée".

Pour remplacer la technologie XSLT dans vos bases de données, 4D vous propose deux solutions :

- utiliser les fonctions équivalentes du module PHP *libxslt*, qui est installé dans 4D depuis la version 14.2. 4D a publié un document spécifique pour vous aider à utiliser le XSL de PHP en remplacement des commandes XSLT de 4D : Télécharger le document "La transformation XSLT avec PHP" (PDF)
- utiliser les possibilités offertes par la commande **TRAITER BALISES 4D**, dont les capacités ont été élargies de manière significative à compter de 4D v14 R4.

XML DECODER

XML DECODER (valeurXML ; var4D)

Paramètre	Type		Description
valeurXML	Texte	➡	Valeur de type texte provenant d'une structure XML
var4D	Champ, Variable	⬅	Variable ou champ 4D devant recevoir la valeur XML convertie

Description

La commande **XML DECODER** convertit une valeur stockée en tant que chaîne XML en une valeur 4D typée. La conversion est effectuée automatiquement en fonction des règles suivantes :

Valeur	Exemples	Conversion sur système français
numérique	<Prix>8,5</Prix><Prix>8.5</Prix>	Réel : 8,5
booléenne	<Double>1</Double> <Double>0</Double> ou <Double>vrai</Double> <Double>faux</Double>	Booléen : Vrai/Faux
BLOB		Décodage base64
Images		Décodage base64 + commande BLOB vers image
Dates	2009-10-25T01:03:20+01:00	!25/10/2009! -> Suppression de la partie heure et du fuseau horaire
Heures	2009-10-25T01:03:20+01:00	?01:03:20? -> Suppression de la partie date. *Attention* : prise en compte du fuseau horaire s'il est différent de celui de l'heure locale. Par exemple "2009-10-25T01:03:20+05:00" donnera ?21:03:20? en heure locale UTC+1

Exemple

Importation de données depuis un document XML dans lequel les valeurs sont stockées en tant qu'attributs.
Exemple de document XML :

```
<CD Date="2003-01-01T00:00:00Z" Description="Ce double CD réédité par EMI en 1995 réunit 4 Stabat mater. Celui de Rossini interprété par l'orchestre Symphonique de Berlin dirigé par Karl Forster. Il est suivi de l'œuvre de Verdi, Philharmonia Orchestra dirigé par Carlo Maria Giulini. Sur le deuxième CD, on trouve Francis Poulenc interprété par Régine Crespin. Cette compilation se termine avec une version moins connue, celle du polonais Karol Szymanowski. Orchestre Symphonique de la Radio Nationale polonaise dirigée par Antoni Wit" Double="true" Duree="7246" Genre="Musique sacrée" ID_CD="5" Interprete="Divers" Prix="8.5" Titre="4 Stabat mater"/>
```

```
Repeter
      MyEvent:=SAX Lire noeud XML(DocRef)

      Au cas ou
      :(MyEvent=XML début élément)
         TABLEAU TEXTE(tAttrNames;0)
         TABLEAU TEXTE(tAttrValues;0)
         SAX LIRE ELEMENT XML(DocRef;vName;vPrefix;tAttrNames;tAttrValues)
         Si(vName="CD")
            CREER ENREGISTREMENT([CD])
            Boucle($i;1;Taille tableau(tAttrNames))
               $attrName:=tAttrNames{$i}
               Au cas ou
               :($attrName="ID_CD")
                  XML DECODER(tAttrValues{$i};[CD]ID_CD)
               :($attrName="Titre")
                  [CD]uvre:=tAttrValues{$i}
               :($attrName="Prix")
                  XML DECODER(tAttrValues{$i};[CD]Prix)
               :($attrName="Date")
                  XML DECODER(tAttrValues{$i};[CD]Date saisie)
               :($attrName="Duree")
                  XML DECODER(tAttrValues{$i};[CD]Durée_totale)
               :($attrName="Double")
                  XML DECODER(tAttrValues{$i};[CD]CD_Double)
               Fin de cas
            Fin de boucle
         Fin de si
      ...
      Fin de cas
Jusque(MyEvent=XML fin document)
```

XML FIXER OPTIONS

XML FIXER OPTIONS (refElément | document ; sélecteur ; valeur {; sélecteur2 ; valeur2 ; ... ; sélecteurN ; valeurN})

Paramètre	Type		Description
refElément | document	Texte, RefDoc	⇒	Référence d'élément XML racine ou Référence de document ouvert
sélecteur	Entier long	⇒	Option à définir
valeur	Entier long	⇒	Valeur de l'option

Description

La commande **XML FIXER OPTIONS** permet de modifier la valeur d'une ou plusieurs option(s) XML pour la structure passée en paramètre.

Cette commande s'applique aux structures XML de type "arbre" (DOM) ou "document" (SAX). Vous pouvez passer en premier paramètre soit une référence d'élément racine (*refElément*), soit une référence de document SAX ouvert (*document*).

Les options définies par cette commande sont utilisées uniquement dans le sens 4D vers XML (elle n'a pas d'effet sur la lecture de valeurs XML dans 4D). Les commandes utilisant ces options sont les suivantes :

- **DOM ECRIRE ATTRIBUT XML**
- **DOM ECRIRE VALEUR ELEMENT XML**
- **SAX AJOUTER VALEUR ELEMENT XML**

Passez dans *sélecteur* l'option à modifier et dans *valeur* la nouvelle valeur de l'option. Vous pouvez passer autant de couples *sélecteur/valeur* que vous souhaitez.

Vous devez utiliser les constantes décrites ci-dessous, placées dans le thème **XML** :

Constante	Type	Valeur	Comment
XML encodage binaire	Entier long	5	Définit la manière dont les données binaires seront converties. **Valeurs possibles :** • XML Base64 (valeur par défaut) : les données binaires sont simplement converties en base64. • XML data URI scheme : les données binaires sont converties en base64 et l'en-tête "data:;base64" est ajouté. Ce format permet principalement à un navigateur de décoder automatiquement une image, et est également requis pour l'insertion d'images . Pour plus d'informations, voir http://en.wikipedia.org/wiki/Data_URI_scheme.
XML encodage chaînes	Entier long	1	Définit la manière dont les chaînes 4D sont converties en valeurs d'éléments. Il ne concerne pas les conversions en attributs pour lesquelles le XML impose l'utilisation de caractères d'échappement. **Valeurs possibles :** • XML avec échappement (valeur par défaut) : conversion des chaînes 4D en valeurs d'éléments XML avec remplacement des caractères. Les données de type texte sont automatiquement analysées de manière à ce que les caractères interdits (<&>') soient remplacés par des entités XML (&<> '"). • XML données brutes : les chaînes 4D sont envoyées en tant que données brutes, 4D n'effectue ni encodage ni analyse. Les valeurs 4D seront converties si possible en fragments XML et insérées en tant qu'enfant de l'élément cible. Si une valeur ne peut pas être considérée comme fragment XML, elle est insérée sous forme de donnée brute dans un nouveau noeud CDATA
XML encodage dates	Entier long	2	Définit la manière dont les dates 4D seront converties. Par exemple, le !01/01/2003! dans le fuseau horaire de Paris. **Valeurs possibles :** • XML ISO (valeur par défaut) : utilisation du format xs:datetime sans indication de fuseau horaire. Résultat : "2003-01-01". La partie heure, si elle est présente dans la valeur 4D (via le SQL), est perdue. • XML local : utilisation du format xs:date avec indication de fuseau horaire. Résultat : "2003-01-01 +01:00". La partie heure, si elle est présente dans la valeur 4D (via le SQL), est perdue. • XML datetime local : utilisation du format xs:dateTime (ISO 8601). Indication de fuseau horaire. Ce format permet de conserver la partie heure, si elle est présente dans la valeur 4D (via le SQL). Résultat : "<Date>2003-01-01T00:00:00 +01:00</Date>". • XML UTC : utilisation du format xs:date. Résultat : "2003-01-01Z". La partie heure, si elle est présente dans la valeur 4D (via le SQL), est perdue. • XML datetime UTC : utilisation du format xs:dateTime (ISO 8601). Ce format permet de conserver la partie heure, si elle est présente dans la valeur 4D (via le SQL). Résultat : "<Date>2003-01-01T00:00:00Z</Date>".
XML encodage heures	Entier long	3	Définit la manière dont les heures 4D seront converties. Par exemple, ?02/00/46? (heure de Paris). L'encodage diffère suivant que vous souhaitez exprimer une heure ou une durée. **Valeurs possibles pour les heures :** • XML datetime UTC : heure exprimée en UTC (Temps Universel Coordonné). A noter que la conversion en UTC est automatique. Résultat : "<Duree>0000-00-00T01:00:46Z</Duree>". • XML datetime local : heure exprimée avec le décalage horaire de la machine du moteur de 4D. Résultat : "<Duree>0000-00-00T02:00:46+01:00</Duree>". • XML datetime local absolu (valeur par défaut) : heure exprimée sans indication de fuseau horaire. Pas de modification de la valeur. Résultat : "<Duree>0000-00-00T02:00:46</Duree>". **Valeurs possibles pour les durées :** • XML secondes : nombre de secondes depuis minuit, pas de modification de la valeur puisqu'elle exprime une durée. Résultat : "<Duree>7246</Duree>". • XML durée : durée exprimée de manière conforme au XML Schema Part 2: Datatypes Second Edition. Pas de modification de la valeur puisqu'elle exprime une durée. Résultat : "<Duree>PT02H00M46S</Duree>".
XML encodage images	Entier long	6	Définit la manière dont les images doivent converties (avant l'encodage en base64). **Valeurs possibles :** • XML convertir en PNG (valeur par défaut) : les images sont converties en PNG avant d'être encodées en base64. • XML codec natif : les images sont converties dans leur premier CODEC natif de stockage avant d'être encodées en base64. Vous devez utiliser ces options pour encoder des images SVG (voir exemple de la commande **XML FIXER OPTIONS**).
XML indentation	Entier long	4	Définit l'indentation du *document* XML. **Valeurs possibles :** • XML avec indentation (valeur par défaut) : le document est indenté. • XML sans indentation : le document n'est pas indenté, son contenu est placé sur une seule ligne.

Notes :

- Les valeurs XML local et XML datetime local ne fournissent pas de dates exprimées en UTC (Temps Universel Coordonné), elles sont converties sans modification mais indiquent le décalage horaire. Ces formats sont utiles dans le cas de conversions successives et réciproques (*round tripping*).
- Les valeurs XML UTC et XML datetime UTC sont équivalentes aux précédentes du point de vue du formatage, mais sont exprimées en UTC. Ces formats sont à privilégier pour assurer l'interopérabilité. Les valeurs ne sont pas modifiées.

Exemple

Insertion d'une image SVG :

```
XML FIXER OPTIONS($refImageElem;XML_encodage_binaire;XML_data_URI_scheme)
XML FIXER OPTIONS($refImageElem;XML_encodage_images;XML_codec_natif)
DOM ECRIRE ATTRIBUT XML($refImageElem;"xlink:href";VarImage)
```

XML LIRE ERREUR

XML LIRE ERREUR (refElément ; texteErreur {; ligne {; colonne}})

Paramètre	Type		Description
refElément	Chaîne	⇒	Référence d'élément XML
texteErreur	Variable	⇐	Texte de l'erreur
ligne	Variable	⇐	Numéro de ligne
colonne	Variable	⇐	Numéro de colonne

Description

La commande **XML LIRE ERREUR** retourne dans le paramètre *texteErreur* la description de l'erreur rencontrée lors du traitement de l'élément XML désigné par le paramètre *refElément*. Les informations retournées sont fournies par la librairie Xerces.dll.

Les paramètres optionnels *ligne* et *colonne* désignent précisément l'emplacement de l'erreur : ils récupèrent respectivement le numéro de la ligne et, dans cette ligne, la position du premier caractère de l'expression à l'origine de l'erreur.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

XML LIRE OPTIONS

XML LIRE OPTIONS (refElément | document ; sélecteur ; valeur {; sélecteur2 ; valeur2 ; ... ; sélecteurN ; valeurN})

Paramètre	Type		Description
refElément | document	Texte, RefDoc	⇒	Référence d'élément XML racine ou Référence de document ouvert
sélecteur	Entier long	⇒	Option à lire
valeur	Entier long	⇐	Valeur courante de l'option

Description

La commande **XML LIRE OPTIONS** permet de lire la valeur courante d'un ou plusieurs paramètre(s) XML définis pour la session courante et l'utilisateur courant.

Passez dans *sélecteur* une des constantes du thème **XML** ci-dessous, indiquant l'option à connaître. La valeur courante de l'option est retournée dans le paramètre *valeur* :

Constante	Type	Valeur	Comment
XML encodage binaire	Entier long	5	Définit la manière dont les données binaires seront converties. **Valeurs possibles :** • XML Base64 (valeur par défaut) : les données binaires sont simplement converties en base64. • XML data URI scheme : les données binaires sont converties en base64 et l'en-tête "data:;base64" est ajouté. Ce format permet principalement à un navigateur de décoder automatiquement une image, et est également requis pour l'insertion d'images . Pour plus d'informations, voir http://en.wikipedia.org/wiki/Data_URI_scheme.
XML encodage chaînes	Entier long	1	Définit la manière dont les chaînes 4D sont converties en valeurs d'éléments. Il ne concerne pas les conversions en attributs pour lesquelles le XML impose l'utilisation de caractères d'échappement. **Valeurs possibles :** • XML avec échappement (valeur par défaut) : conversion des chaînes 4D en valeurs d'éléments XML avec remplacement des caractères. Les données de type texte sont automatiquement analysées de manière à ce que les caractères interdits (<&>') soient remplacés par des entités XML (&<> '"). • XML données brutes : les chaînes 4D sont envoyées en tant que données brutes, 4D n'effectue ni encodage ni analyse. Les valeurs 4D seront converties si possible en fragments XML et insérées en tant qu'enfant de l'élément cible. Si une valeur ne peut pas être considérée comme fragment XML, elle est insérée sous forme de donnée brute dans un nouveau noeud CDATA
XML encodage dates	Entier long	2	Définit la manière dont les dates 4D seront converties. Par exemple, le !01/01/2003! dans le fuseau horaire de Paris. **Valeurs possibles :** • XML ISO (valeur par défaut) : utilisation du format xs:datetime sans indication de fuseau horaire. Résultat : "2003-01-01". La partie heure, si elle est présente dans la valeur 4D (via le SQL), est perdue. • XML local : utilisation du format xs:date avec indication de fuseau horaire. Résultat : "2003-01-01 +01:00". La partie heure, si elle est présente dans la valeur 4D (via le SQL), est perdue. • XML datetime local : utilisation du format xs:dateTime (ISO 8601). Indication de fuseau horaire. Ce format permet de conserver la partie heure, si elle est présente dans la valeur 4D (via le SQL). Résultat : "<Date>2003-01-01T00:00:00 +01:00</Date>". • XML UTC : utilisation du format xs:date. Résultat : "2003-01-01Z". La partie heure, si elle est présente dans la valeur 4D (via le SQL), est perdue. • XML datetime UTC : utilisation du format xs:dateTime (ISO 8601). Ce format permet de conserver la partie heure, si elle est présente dans la valeur 4D (via le SQL). Résultat : "<Date>2003-01-01T00:00:00Z</Date>".
XML encodage heures	Entier long	3	Définit la manière dont les heures 4D seront converties. Par exemple, ?02/00/46? (heure de Paris). L'encodage diffère suivant que vous souhaitez exprimer une heure ou une durée. **Valeurs possibles pour les heures :** • XML datetime UTC : heure exprimée en UTC (Temps Universel Coordonné). A noter que la conversion en UTC est automatique. Résultat : "<Duree>0000-00-00T01:00:46Z</Duree>". • XML datetime local : heure exprimée avec le décalage horaire de la machine du moteur de 4D. Résultat : "<Duree>0000-00-00T02:00:46+01:00</Duree>". • XML datetime local absolu (valeur par défaut) : heure exprimée sans indication de fuseau horaire. Pas de modification de la valeur. Résultat : "<Duree>0000-00-00T02:00:46</Duree>". **Valeurs possibles pour les durées :** • XML secondes : nombre de secondes depuis minuit, pas de modification de la valeur puisqu'elle exprime une durée. Résultat : "<Duree>7246</Duree>". • XML durée : durée exprimée de manière conforme au XML Schema Part 2: Datatypes Second Edition. Pas de modification de la valeur puisqu'elle exprime une durée. Résultat : "<Duree>PT02H00M46S</Duree>".
XML encodage images	Entier long	6	Définit la manière dont les images doivent converties (avant l'encodage en base64). **Valeurs possibles :** • XML convertir en PNG (valeur par défaut) : les images sont converties en PNG avant d'être encodées en base64. • XML codec natif : les images sont converties dans leur premier CODEC natif de stockage avant d'être encodées en base64. Vous devez utiliser ces options pour encoder des images SVG (voir exemple de la commande **XML FIXER OPTIONS**).
XML indentation	Entier long	4	Définit l'indentation du *document* XML. **Valeurs possibles :** • XML avec indentation (valeur par défaut) : le document est indenté. • XML sans indentation : le document n'est pas indenté, son contenu est placé sur une seule ligne.

_o_XSLT APPLIQUER TRANSFORMATION

_o_XSLT APPLIQUER TRANSFORMATION (sourceXML ; feuilleXSL ; résultat {; compileFeuille})

Paramètre	Type		Description
sourceXML	Chaîne, BLOB	⇒	Nom ou chemin d'accès du document XML source ou BLOB contenant le XML source
feuilleXSL	Chaîne, BLOB	⇒	Nom ou chemin d'accès du document contenant la feuille de style XSL, ou BLOB contenant la feuille de style XSL
résultat	Chaîne, BLOB	⇒	Nom ou chemin d'accès du document recevant le résultat de la transformation XSLT, ou BLOB recevant le résultat de la transformation XSLT
compileFeuille	Booléen	⇒	Vrai : optimise la transformation XSLT Faux ou omis : pas d'optimisation, efface le fichier XSL compilé s'il existe

Note de compatibilité

A compter de 4D v14 R4, les commandes de transformation XSL sont obsolètes. Par compatibilité, elles sont toujours prises en charge dans 4D, mais leur usage est désormais déconseillé. L'utilisation de la technologie XSLT ne sera plus possible dans les versions futures de 4D. Pour plus d'informations, veuillez vous reporter à la section **Présentation des commandes XML génériques**.

_o_XSLT FIXER PARAMETRE

_o_XSLT FIXER PARAMETRE (nomParam ; valeurParam)

Paramètre	Type		Description
nomParam	Chaîne	⇨	Nom du paramètre à chercher dans la feuille XSL
valeurParam	Chaîne	⇨	Valeur du paramètre à utiliser dans le document transformé

Note de compatibilité

A compter de 4D v14 R4, les commandes de transformation XSL sont obsolètes. Par compatibilité, elles sont toujours prises en charge dans 4D, mais leur usage est désormais déconseillé. L'utilisation de la technologie XSLT ne sera plus possible dans les versions futures de 4D. Pour plus d'informations, veuillez vous reporter à la section **Présentation des commandes XML génériques**.

_o_XSLT LIRE ERREUR

_o_XSLT LIRE ERREUR (texteErreur {; ligne {; colonne}})

Paramètre	Type		Description
texteErreur	Variable	⇦	Texte de l'erreur
ligne	Variable	⇦	Numéro de ligne
colonne	Variable	⇦	Numéro de colonne

Note de compatibilité

A compter de 4D v14 R4, les commandes de transformation XSL sont obsolètes. Par compatibilité, elles sont toujours prises en charge dans 4D, mais leur usage est désormais déconseillé. L'utilisation de la technologie XSLT ne sera plus possible dans les versions futures de 4D. Pour plus d'informations, veuillez vous reporter à la section **Présentation des commandes XML génériques**.

XML DOM

- Présentation des commandes XML DOM
- DOM Ajouter element XML
- DOM Ajouter noeud enfant XML
- DOM Analyser source XML
- DOM Analyser variable XML
- DOM Chercher element XML
- DOM Chercher element XML par ID
- DOM Compter attributs XML
- DOM Compter elements XML
- DOM Creer element XML
- DOM Creer element XML tableaux
- DOM Creer ref XML
- DOM ECRIRE ATTRIBUT XML
- DOM ECRIRE NOM ELEMENT XML
- DOM ECRIRE VALEUR ELEMENT XML
- DOM EXPORTER VERS FICHIER
- DOM EXPORTER VERS VARIABLE
- DOM FERMER XML
- DOM FIXER DECLARATION XML
- DOM Inserer element XML
- DOM LIRE ATTRIBUT XML PAR INDEX
- DOM LIRE ATTRIBUT XML PAR NOM
- DOM Lire dernier element XML enfant
- DOM Lire element XML
- DOM Lire element XML frere precedent
- DOM Lire element XML frere suivant
- DOM Lire element XML parent
- DOM Lire element XML racine
- DOM Lire informations XML
- DOM LIRE NOEUDS ENFANTS XML
- DOM LIRE NOM ELEMENT XML
- DOM Lire premier element XML enfant
- DOM Lire ref document XML
- DOM LIRE VALEUR ELEMENT XML
- DOM SUPPRIMER ATTRIBUT XML
- DOM SUPPRIMER ELEMENT XML

Présentation des commandes XML DOM

4D inclut un ensemble de commandes permettant d'écrire et d'analyser des objets contenant des données XML (eXtensible Markup Language).

Note à propos du mode préemptif : Les références XML créées par un process préemptif peuvent être utilisées dans ce process uniquement. A l'inverse, les références XML créées par des process coopératifs peuvent être utilisées par tout autre process coopératif, mais ne peuvent pas être utilisées par un process préemptif.

A propos du langage XML

Le langage XML est une norme d'échange de données. Il est basé sur l'emploi de balises permettant de décrire de manière précise les données échangées ainsi que leur structure. Les fichiers XML sont des fichiers au format Texte, leur contenu est analysé (parsing) par les applications qui importent les données. Aujourd'hui, de nombreuses applications prennent en charge ce format.
Pour plus d'informations sur le XML, reportez-vous, par exemple, au site *http://xmlfr.org*.

Pour la prise en charge du XML, 4D utilise une librairie nommée Xerces.dll développée par la société Apache Foundation. 4D prend en charge XML version 1.0.

Note : 4D permet également d'importer et d'exporter directement des données au format XML à l'aide de l'éditeur d'import/export standard.

DOM et SAX

Les commandes de ce thème sont préfixées DOM. En effet, 4D propose deux ensembles distincts de commandes XML: les commandes DOM (Document Object Model) et SAX (Simple API XML), qui constituent deux modes d'analyse différents des documents XML.

- Le mode DOM effectue l'analyse d'une source XML et construit sa structure (son "arbre") en mémoire. De ce fait, l'accès à chaque élément de la source est extrêmement rapide. Cependant, la totalité de l'arbre étant contenu dans la mémoire, le traitement de gros documents XML peut dépasser la capacité de la mémoire et provoquer des erreurs.
- Le mode SAX ne construit pas d'arbre en mémoire. Dans ce mode, des "événements" (tels que le début et la fin d'un élément) sont générés lors de l'analyse de la source. Ce mode autorise l'analyse de documents XML de toute taille, quelle que soit la quantité de mémoire disponible. Les commandes SAX sont regroupées dans le thème "**XML SAX**". Pour plus d'informations, reportez-vous à la section **Présentation des commandes XML SAX**.

Pour plus d'informations sur les standards XML, vous pouvez consulter les sites http://www.saxproject.org/?selected=event et http://www.w3schools.com/xml/.

Création, ouverture et fermeture des documents XML via DOM

Les objets créés, modifiés ou analysés par les commandes DOM de 4D peuvent être des textes, des URLs, des documents ou des BLOBs. Les commandes DOM utilisées pour l'ouverture des objets XML dans 4D sont **DOM Analyser source XML** et **DOM Analyser variable XML**.
De nombreuses commandes permettent ensuite de lire, d'analyser et d'écrire les éléments et les attributs. La récupération des erreurs s'effectue via la commande **DOM Analyser variable XML** (commune aux deux standards XML).
La commande **XML LIRE ERREUR** permet de finalement refermer la source.

Note sur l'usage de paramètres BLOBs XML: Les structures XML sont basées sur des données de type texte, il est recommandé de les manipuler via des variables ou des champs de type Texte. Pour des raisons historiques, les commandes XML de 4D (par exemple **DOM Analyser variable XML**) acceptent des paramètres de type BLOBs. En effet, dans les versions précédentes de 4D, la taille des variables de type Texte était limitée à 32 Ko. Depuis la version 11 de 4D, les variables et champs texte peuvent contenir jusqu'à 2 Go de données. La limitation d'origine étant supprimée, il est désormais fortement déconseillé de stocker des textes dans des BLOBs. L'usage de BLOBs est à réserver au traitement de données binaires. Par conformité aux spécifications du XML, à compter de 4D v12 les données binaires sont automatiquement encodées en Base64, même si le BLOB contient du texte.

Utilisation de la notation XPath

Trois commandes XML DOM (**DOM Creer element XML**, **DOM Chercher element XML** et **DOM ECRIRE VALEUR ELEMENT XML**) acceptent la notation XPath pour l'accès aux éléments XML.
La notation XPath est issue du langage XPath, consacré à la navigation à l'intérieur des structures XML. Elle permet de désigner directement des éléments au sein d'une structure XML via une syntaxe du type "chemin d'accès", sans devoir nécessairement indiquer le chemin complet pour y parvenir. Soit par exemple la structure suivante :

```
<RootElement>       <Elem1>          <Elem2>             <Elem3 Font=Verdana Size=10> </Elem3>
      </Elem2>       </Elem1>    </RootElement>
```

La notation XPath permet d'accéder à l'élément 3 via la syntaxe */RootElement/Elem1/Elem2/Elem3*.

4D accepte également les éléments XPath **indexés**, avec la syntaxe **Elément[NumElément]**. Soit par exemple la structure suivante :

```
<RootElement>       <Elem1>          <Elem2>aaa</Elem2>          <Elem2>bbb</Elem2>
      <Elem2>ccc</Elem2>       </Elem1>    </RootElement>
```

La notation XPath permet d'accéder à la valeur "ccc" via la syntaxe */RootElement/Elem1/Elem2[3]*.

Pour une illustration de la notation XPath, reportez-vous aux exemples des commandes **DOM Creer element XML** et **DOM Chercher element XML**.

Jeux de caractères

Les jeux de caractères suivants sont pris en charge par les commandes XML DOM et XML SAX de 4D :

- ASCII
- UTF-8
- UTF-16 (Big/Small Endian)
- UCS4 (Big/Small Endian)
- EBCDIC code pages IBM037, IBM1047 and IBM1140 encodings,
- ISO-8859-1 (ou Latin1)
- Windows-1252.

Terminologie

Le langage XML utilise de nombreux termes et acronymes spécifiques. Cette liste non exhaustive explicite les principales notions XML utilisées par les commandes et fonctions de 4D.

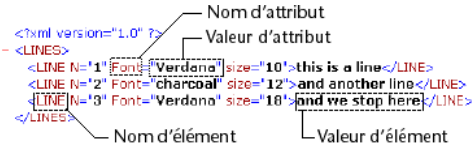
Attribut : Sous-balise XML associée à un élément. Un attribut comporte toujours un nom et une valeur (cf. schéma ci-dessous).

Bien formé : Un document XML est déclaré “bien formé” par l’analyseur XML lorsqu’il est conforme aux spécifications XML génériques. Voir aussi Validation.

DTD : Document Type Declaration (Déclaration de type de document). La DTD recense l’ensemble des règles et des propriétés spécifiques que doit suivre un document XML. Ces règles définissent notamment le nom et le contenu de chaque balise ainsi que leur contexte. Cette formalisation des éléments permet de vérifier qu’un document XML est conforme (dans ce cas il est déclaré “valide”).
La DTD peut être incluse dans le document XML (DTD interne) ou dans un document tiers (DTD externe). A noter que la DTD n’est pas obligatoire.

Elément : Balise XML. Un élément comporte toujours un nom et une valeur. Facultativement, un élément peut contenir des attributs (cf. schéma).

Eléments et attributs



Enfant : Dans une structure XML, élément d’un niveau directement inférieur à un autre.

Frère : Dans une structure XML, élément du même niveau qu'un autre.

Parent : Dans une structure XML, élément d’un niveau directement supérieur à un autre.

Parsing, **parser** (**Analyser**, **analyseur**) : Action d’analyser le contenu d’un objet structuré afin d’en extraire les informations utiles. Les commandes du thème “XML” permettent d’analyser le contenu de tout objet XML.

Racine (**Root**) : Elément situé au premier niveau d’une structure XML.

RefElément : Référence XML utilisée par les commandes XML de 4D pour désigner une structure XML (documents ou élément). Cette référence est constituée de 8 caractères codés sous forme hexadécimale, ce qui signifie que sa longueur est de 16 ou 32 caractères selon que vous utilisez un système 32 ou 64 bits. Il est conseillé de déclarer les références XML à l'aide de la directive **C_TEXTE**.

Structure XML : objet XML structuré. Cet objet peut être un document, une variable, un élément.

Validation : Un document XML est “validé” par l’analyseur XML lorsqu’il est “bien formé” et conforme aux spécifications de la DTD. Voir aussi Bien formé.

XML : eXtensible Markup Language (Langage balisé évolutif). Norme d’échange de données informatisées permettant de transférer des données ainsi que leur structure. Le langage XML est basé sur l’emploi de balises et d’une syntaxe spécifiques, à l’instar du langage HTML. Toutefois, à la différence de ce dernier, le langage XML permet de définir des balises personnalisées.

XSL : eXtensible Stylesheet Language (Langage des feuilles de style évolutif). Langage permettant de définir des feuilles de style utilisables pour traiter et afficher le contenu d’un document XSL.

Gestion des erreurs

De nombreuses fonctions de ce thème retournent une référence d'élément XML. Si une erreur se produit durant l'exécution d'une fonction (par exemple si la référence de l’élément racine est invalide), la variable OK prend la valeur 0 et une erreur est générée.

De plus, la référence retournée dans ce cas est une suite de caractères "0" (16 caractères en 32 bits, ou 32 caractères en 64 bits).

DOM Ajouter element XML

DOM Ajouter element XML (refElémentCible ; refElémentSource) -> Résultat

Paramètre	Type		Description
refElémentCible	Texte	⇒	Référence de l'élément XML parent
refElémentSource	Texte	⇒	Référence de l'élément XML à ajouter
Résultat	Texte	⇐	Référence du nouvel élément XML

Description

La commande **DOM Ajouter element XML** permet d'ajouter un nouvel élément XML aux enfants de l'élément XML dont la référence est passée dans le paramètre *refElémentCible*.

Passez dans *refElémentSource* l'élément à ajouter. Cet élément doit être passé en tant que référence d'un élément XML existant dans un arbre DOM. Il est ajouté après le dernier élément enfant existant de *refElémentCible*.

Exemple

Voir l'exemple de la commande **DOM Inserer element XML**.

DOM Ajouter noeud enfant XML

DOM Ajouter noeud enfant XML (refElément ; typeEnfant ; valeurEnfant) -> Résultat

Paramètre	Type		Description
refElément	Texte	⇒	Référence d'élément XML
typeEnfant	Entier long	⇒	Type d'enfant à ajouter
valeurEnfant	Texte, BLOB	⇒	Texte ou variable (Texte ou BLOB) dont la valeur doit être insérée en tant que noeud enfant
Résultat	Texte	⇐	Référence de l'élément XML enfant

Description

La commande **DOM Ajouter noeud enfant XML** permet d'ajouter la valeur *valeurEnfant* au noeud XML désigné par *refElément*.

Le type de noeud créé est défini par le paramètre *typeEnfant*. Passez dans ce paramètre l'une des constantes suivantes, placées dans le thème **XML** :

Constante	Type	Valeur
XML CDATA	Entier long	7
XML commentaire	Entier long	2
XML DOCTYPE	Entier long	10
XML Donnée	Entier long	6
XML Elément	Entier long	11
XML instruction de traitement	Entier long	3

Passez dans *valeurEnfant* les données à insérer. Vous pouvez passer une chaîne ou une variable 4D (chaîne ou BLOB). Le contenu de ce paramètre sera toujours converti en texte.

Note : Si le paramètre *refElément* désigne le noeud Document (noeud de plus haut niveau), la commande insère un noeud "Doctype" avant tout autre noeud. Il en va de même pour les instructions de traitement et les commentaires, qui sont toujours insérés avant le noeud racine (mais après le noeud Doctype).

Exemple 1

Ajout d'un noeud de type texte :

```
Reference:=DOM Creer element XML(refElement;"monElement")
DOM ECRIRE VALEUR ELEMENT XML(Reference ;"Bonjour")
temp:=DOM Creer element XML(Reference ;"br")
temp:=DOM Ajouter noeud enfant XML(Reference;XML Donnée;"La")
temp:=DOM Creer element XML(Reference;"br")
temp:=DOM Ajouter noeud enfant XML(Reference;XML Donnée;"France")
```

Résultat :

```
<monElement>Bonjour<br/>La<br/>France</monElement>
```

Exemple 2

Ajout d'un noeud de type instruction de traitement :

```
$Txt_instruction:="xml-stylesheet type = \"text/xsl\" href=\"style.xsl\""
Reference:=DOM Ajouter noeud enfant XML(refElement;XML instruction de traitement;$Txt_instruction )
```

Résultat (inséré avant le premier élément) :

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

Exemple 3

Ajout d'un noeud de type commentaire :

```
Reference:=DOM Ajouter noeud enfant XML(refElement;XML commentaire;"Hello world")
```

Résultat :

```
<!--Hello world-->
```

Exemple 4

Ajout d'un noeud de type CDATA :

```
Reference:=DOM Ajouter noeud enfant XML(refElement;XML CDATA;"12 < 18")
```

Résultat :

```
<element><![CDATA[12 < 18]]></element>
```

Exemple 5

Ajout ou remplacement d'un noeud de type déclaration Doctype :

```
Reference:=DOM Ajouter noeud enfant XML(refElement;XML DOCTYPE;"Books SYSTEM \"Book.DTD\"")
```

Résultat (inséré avant le premier élément) :

```
<!DOCTYPE Books SYSTEM "Book.DTD">
```

Exemple 6

Ajout ou remplacement d'un noeud de type Elément.

- si le paramètre *valeurEnfant* est un fragment XML, il est inséré en tant que noeuds enfants :

  ```
  Reference:=DOM Ajouter noeud enfant XML(refElement;XML Elément;"<child>simon</child><child>eva</child>")
  ```

  Résultat :

  ```
  <parent>     <child>simon</child>     <child>eva</child> </parent>
  ```

- sinon, un nouvel élément enfant vide est ajouté :

  ```
  Reference:=DOM Ajouter noeud enfant XML(refElement;XML Elément;"tbreak")
  ```

  Résultat :

  ```
  <parent>     <tbreak/> </parent>
  ```

Si le contenu de *valeurEnfant* est invalide, une erreur est retournée.

DOM Analyser source XML

DOM Analyser source XML (nomFichier {; validation {; dtd | schéma}}) -> Résultat

Paramètre	Type		Description
nomFichier	Chaîne	⇒	Chemin d'accès du document
validation	Booléen	⇒	Vrai = Validation, Faux = Pas de validation
dtd | schéma	Chaîne	⇒	Emplacement de la DTD ou du schéma XML
Résultat	Chaîne	⇐	Référence de l'élément XML (16 caractères)

Description

La commande **DOM Analyser source XML** analyse un document contenant une structure XML et retourne une référence pour ce document. La commande peut valider ou non le document via une DTD ou un schéma XML (document XSD, XML Schema Definition).
Le document peut être situé sur disque ou sur Internet/Intranet.

Note : L'exécution de la commande **DOM Analyser source XML** est synchrone.

Vous pouvez passer dans le paramètre *document* :

- soit un chemin d'accès complet standard (du type C:\\Dossier\\Fichier\\... sous Windows et MacintoshHD:Dossier:Fichier sous Mac OS),
- soit un chemin Unix sous Mac OS (débutant obligatoirement par /).
- soit un chemin réseau du type http://www.site.com/Fichier ou ftp://public.ftp.com...

Le paramètre booléen *validation* vous permet d'indiquer si vous souhaitez que la structure soit validée ou non.

- Si *validation* vaut Vrai, la structure sera validée. Dans ce cas, l'analyseur tentera de valider la structure XML du document sur la base de la référence DTD ou XSD incluse dans le document, ou via la DTD ou le schéma XML désigné(e) par le troisième paramètre s'il est passé.
- Si *validation* vaut Faux, la structure ne sera pas validée.

Si vous passez Vrai dans *validation* et omettez le troisième paramètre, la commande tentera de valider la structure XML via une référence DTD ou XSD trouvée dans la structure elle-même. La validation peut être indirecte : si la structure contient une référence vers une fichier DTD qui lui-même contient une référence vers un fichier XSD, la commande tentera d'effectuer les deux validations.

Le troisième paramètre vous permet de désigner une DTD spécifique ou un schéma XML pour l'analyse du document. Si vous utilisez ce paramètre, la commande ne tient pas compte de la DTD référencée dans le document XML.

Validation par DTD
Il existe deux moyens pour désigner une DTD :

- en tant que référence. Il vous suffit pour cela de passer le chemin d'accès complet de la nouvelle DTD (extension "dtd") dans le paramètre *dtd*. Si le document désigné ne contient pas de DTD valide, le paramètre *dtd* est ignoré et une erreur est générée.
- directement dans un texte. Dans ce cas, si le contenu du paramètre débute par "<?xml", 4D considérera qu'il s'agit de la DTD ; dans le cas contraire, 4D considérera qu'il s'agit d'un chemin d'accès.

Validation par schema
Pour valider le document via un schéma XML, il suffit de passer dans le troisième paramètre un fichier ou un URL d'extension"xsd" au lieu de "dtd". La validation par schéma XML est considérée comme plus souple et plus puissante que la validation par DTD. Le langage des documents XSD est basé sur le langage XML. Les schémas XML prennent notamment en charge des types de données. Pour plus d'informations sur les schémas XML, reportez-vous à l'adresse *http://www.w3.org/XML/Schema*.

Si la validation ne peut être effectuée (pas de DTD ou d'XSD, URL incorrect, etc.), une erreur est générée. La variable système Error indique le numéro de l'erreur. Vous pouvez intercepter cette erreur à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

La commande retourne une chaîne de 16 caractères (RefElément) constituant la référence en mémoire de la structure virtuelle du document. Cette référence devra être utilisée avec les autres commandes d'analyse XML.

Important : Une fois que vous n'en avez plus besoin, n'oubliez pas d'appeler la commande **DOM FERMER XML** avec cette référence afin de libérer la mémoire.

Exemple 1

Ouverture sans validation d'un document XML situé sur disque :

```
$ref_XML_Struct:=DOM Analyser source XML("C:\\import.xml")
```

Exemple 2

Ouverture sans validation d'un document XML situé à côté du fichier de structure de la base :

```
$ref_XML_Struct:=DOM Analyser source XML("import.xml")
```

Exemple 3

Ouverture d'un document XML situé sur disque et validation à l'aide d'une DTD située sur le disque :

```
$ref_XML_Struct:=DOM Analyser source XML("C:\\import.xml";Vrai;"C:\\import_dtd.xml")
```

Exemple 4

Ouverture sans validation d'un document XML situé à un URL spécifique :

```
$ref_XML_Struct:=DOM Analyser source XML("http://www.4D.fr/xml/import.xml")
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

DOM Analyser variable XML

DOM Analyser variable XML (variable {; validation {; dtd | schéma}}) -> Résultat

Paramètre	Type		Description
variable	BLOB, Texte	⇒	Nom de la variable
validation	Booléen	⇒	Vrai = Validation, Faux = Pas de validation
dtd | schéma	Chaîne	⇒	Emplacement de la DTD ou du schéma XML
Résultat	Chaîne	⇐	Référence de l'élément XML (16 caractères)

Description

La commande **DOM Analyser variable XML** analyse une variable de type BLOB ou Texte contenant une structure XML et retourne une référence pour cette variable. La commande peut valider ou non la structure via une DTD ou un schéma XML (document XSD, XML Schema Definition). le document.

Passez dans le paramètre *variable* le nom de la variable BLOB ou Texte contenant l'objet XML.

Le paramètre booléen *validation* vous permet d'indiquer si vous souhaitez que la structure soit validée ou non.

- Si *validation* vaut Vrai, la structure sera validée. Dans ce cas, l'analyseur tentera de valider la structure XML du document sur la base de la référence DTD ou XSD incluse dans le document, ou via la DTD ou le schéma XML désigné(e) par le troisième paramètre s'il est passé.
- Si *validation* vaut Faux, la structure ne sera pas validée.

Si vous passez Vrai dans *validation* et omettez le troisième paramètre, la commande tentera de valider la structure XML via une référence DTD ou XSD trouvée dans la structure elle-même. La validation peut être indirecte : si la structure contient une référence vers une fichier DTD qui lui-même contient une référence vers un fichier XSD, la commande tentera d'effectuer les deux validations.

Le troisième paramètre vous permet de désigner une DTD spécifique ou un schéma XML pour l'analyse du document. Si vous utilisez ce paramètre, la commande ne tient pas compte de la DTD référencée dans le document XML.

Validation par DTD
Il existe deux moyens pour désigner une DTD :

- en tant que référence. Il vous suffit pour cela de passer le chemin d'accès complet de la nouvelle DTD (extension "dtd") dans le paramètre *dtd*. Si le document désigné ne contient pas de DTD valide, le paramètre *dtd* est ignoré et une erreur est générée.
- directement dans un texte. Dans ce cas, si le contenu du paramètre débute par "<?xml", 4D considérera qu'il s'agit de la DTD ; dans le cas contraire, 4D considérera qu'il s'agit d'un chemin d'accès.

Validation par schema
Pour valider le document via un schéma XML, il suffit de passer dans le troisième paramètre un fichier ou un URL d'extension"xsd" au lieu de "dtd". La validation par schéma XML est considérée comme plus souple et plus puissante que la validation par DTD. Le langage des documents XSD est basé sur le langage XML. Les schémas XML prennent notamment en charge des types de données. Pour plus d'informations sur les schémas XML, reportez-vous à l'adresse *http://www.w3.org/XML/Schema*.

Si la validation ne peut être effectuée (pas de DTD ou d'XSD, URL incorrect, etc.), une erreur est générée. La variable système Error indique le numéro de l'erreur. Vous pouvez intercepter cette erreur à l'aide d'une méthode installée par la commande **APPELER SUR ERREUR**.

La commande retourne une chaîne de caractères (*RefElément*) constituant la référence en mémoire de la structure virtuelle de la variable. Cette référence devra être utilisée avec les autres commandes d'analyse XML.

Important : Une fois que vous n'en avez plus besoin, n'oubliez pas d'appeler la commande **DOM FERMER XML** avec cette référence afin de libérer la mémoire.

Exemple 1

Ouverture sans validation d'un objet XML situé dans une variable Texte 4D :

```
C_TEXTE(maVarTexte)
C_HEURE(vDoc)
C_TEXTE($ref_XML_Struct)

vDoc:=Ouvrir document("Document.xml")
Si(OK=1)
   RECEVOIR PAQUET(vDoc;maVarTexte;32000)
   FERMER DOCUMENT(vDoc)
   $ref_XML_Struct:=DOM Analyser variable XML(maVarTexte)
Fin de si
```

Exemple 2

Ouverture sans validation d'un document XML situé dans un BLOB 4D :

```
C_BLOB(maVarBlob)
C_TEXTE($ref_XML_Struct)

DOCUMENT VERS BLOB("c:\\import.xml";maVarBlob)
$ref_XML_Struct:=DOM Analyser variable XML(maVarBlob)
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

DOM Chercher element XML

DOM Chercher element XML (refElément ; xChemin {; tabRefEléments}) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML
xChemin	Texte	➡	Chemin XPath de l'élément à chercher
tabRefEléments	Tableau chaîne	⬅	Liste des références d'éléments trouvés (le cas échéant)
Résultat	Chaîne	➲	Référence de l'élément trouvé (le cas échéant)

Description

La commande **DOM Chercher element XML** vous permet de rechercher des éléments XML spécifiques dans une structure XML. La recherche débute à l'élément désigné par le paramètre *refElément*.

Le noeud XML à chercher est défini par le paramètre *xChemin*, exprimé en notation XPath (cf. paragraphe "Utilisation de la notation XPath" dans la section **Présentation des commandes XML DOM**). Il est possible d'utiliser des éléments indexés.

Note : Conformément à la norme XML, la recherche différencie les majuscules et les minuscules.

La commande retourne en résultat la référence XML de l'élément trouvé.

Lorsque le tableau chaîne *tabRefEléments* est passé, la commande le remplit avec la liste des références XML trouvées. Dans ce cas, la commande retourne en résultat le premier élément du tableau *tabRefEléments*. Ce paramètre est utile lorsque plusieurs éléments de même nom existent à l'emplacement désigné par le paramètre *xChemin*.

Exemple 1

Cet exemple permet de rechercher rapidement un élément XML et d'afficher sa valeur :

```
vTrouvé:=DOM Chercher element XML(vRefElem;"Items/Book[15]/Title")
DOM LIRE VALEUR ELEMENT XML(vTrouvé;valeur)
ALERTE("La valeur de l'élément est : \""+valeur+"\"")
```

La même recherche peut également être effectuée ainsi :

```
vTrouvé:=DOM Chercher element XML(vRefElem;"Items/Book[15]")
vTrouvé:=DOM Chercher element XML(vTrouvé;"Book/Title")
DOM LIRE VALEUR ELEMENT XML(vTrouvé;valeur)
ALERTE("La valeur de l'élément est : \""+valeur+"\"")
```

Note : Comme vous pouvez le constater dans l'exemple ci-dessus, le chemin XPath doit toujours débuter par le nom de l'élément courant. Cette précision est importante lorsque vous manipulez des chemins XPath relatifs.

Exemple 2

Soit la structure XML suivante :

```
<Racine>    <Elem1>       <Elem2>aaa</Elem2>       <Elem2>bbb</Elem2>       <Elem2>ccc</Elem2>    </Elem1>
</Racine>
```

Le code suivant permet de récupérer la référence de chaque élément Elem2 dans le tableau tAtrouvés :

```
TABLEAU TEXTE(tAtrouvés;0)
vTrouvé:=DOM Chercher element XML(vRefElem;"/Racine/Elem1/Elem2";tAtrouvés)
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

Une erreur est générée lorsque :

- la référence de l'élément n'est pas valide
- le chemin xPath passé n'est pas valide.

DOM Chercher element XML par ID

DOM Chercher element XML par ID (refElément ; id) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
id	Chaîne	→	Valeur de l'attribut ID de l'élément à chercher
Résultat	Chaîne	←	Référence de l'élément trouvé (le cas échéant)

Description

La commande **DOM Chercher element XML par ID** vous permet de rechercher, à l'intérieur d'un document XML, l'élément dont l'attribut id est égal à la valeur passée dans le paramètre *id*.

Passez dans *refElément* la référence d'un élément du document XML dans lequel vous souhaitez effectuer la recherche. Vous pouvez passer la référence de l'élément racine ou de tout autre élément, la recherche ne tient pas compte de la position de *refElément* et s'effectue toujours dans la totalité du document.

La commande retourne en résultat la référence XML de l'élément trouvé.

Attention : En XML, l'attribut id permet d'associer un identifiant unique à chaque élément d'un document. La valeur de l'attribut id doit être un nom XML valide et doit être unique dans le document XML, tous éléments confondus (contrainte de validité). Le bon fonctionnement de la commande **DOM Chercher element XML par ID** requiert que cette contrainte soit respectée, sinon le résultat sera aléatoire (la commande retournera la référence du premier élément trouvé dans le document).

DOM Compter attributs XML

DOM Compter attributs XML (refElément) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML
Résultat	Entier long	⮌	Nombre d'attributs

Description

La commande **DOM Compter attributs XML** retourne le nombre d'attributs XML présents dans l'élément XML désigné par *refElément*. Pour plus d'informations sur les attributs XML, reportez-vous à la section **Présentation des commandes XML DOM**.

Exemple

Avant de récupérer les valeurs des éléments dans un tableau, vous souhaitez connaître le nombre d'attributs dans l'élément XML suivant :

```
<?xml version="1.0" ?>
- <LINES>
    <LINE N="1" Font="Verdana" size="10">this is a line</LINE>
    <LINE N="2" Font="charcoal" size="12">and another line</LINE>
    <LINE N="3" Font="Verdana" size="18">and we stop here</LINE>
  </LINES>
```

```
C_BLOB(maVarBlob)
C_TEXTE($ref_XML_Parent;$ref_XML_Enfant)
C_TEXTE(monRésultat)
C_ENTIER LONG($nbAttributs)

$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire premier element XML enfant($ref_XML_Parent)

$nbAttributs:=DOM Compter attributs XML($ref_XML_Enfant)
TABLEAU TEXTE(tAttrib;$nbAttributs)
TABLEAU TEXTE(tValAttrib;$nbAttributs)
Boucle($i;1;$nbAttributs)
   DOM LIRE ATTRIBUT XML PAR INDEX($ref_XML_Enfant;$i;tAttrib{$i};tValAttrib{$i})
Fin de boucle
```

Dans l'exemple ci-dessus, $nbAttributs vaut 3, tAttrib{1} contient "Font", tAttrib{2} contient "N" et tAttrib{3} contient "size". tValAttrib contient "Verdana", "1" et "10".

Note : Le numéro d'indice ne correspond pas à l'emplacement de l'attribut dans le fichier XML affiché sous forme de texte. En XML, l'indice d'un attribut indique sa position parmi les attributs classés par ordre alphabétique (en fonction de leur nom).

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM Compter elements XML

DOM Compter elements XML (refElément ; nomElément) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	⇒	Référence d'élément XML
nomElément	Chaîne	⇒	Nom d'éléments XML à compter
Résultat	Entier long	⇐	Nombre d'éléments

Description

La commande **DOM Compter elements XML** retourne le nombre d'éléments "enfants" dépendants de l'élément parent *refElément* et nommés *nomElément*.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM Creer element XML

DOM Creer element XML (refElément ; xChemin {; nomAttribut ; valeurAttribut} {; nomAttribut2 ; valeurAttribut2 ; ... ; nomAttributN ; valeurAttributN}) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML racine
xChemin	Texte	➡	Chemin XPath de l'élément XML à créer
nomAttribut	Chaîne	➡	Attribut à définir
valeurAttribut	Chaîne, Booléen, Entier long, Réel, Heure, Date	➡	Nouvelle valeur d'attribut
Résultat	Chaîne	⮌	Référence de l'élément XML créé

Description

La commande **DOM Creer element XML** permet de créer un nouvel élément dans l'élément XML *refElément* à l'emplacement du noeud désigné par le paramètre *xChemin* et de lui ajouter éventuellement des attributs.

Passez dans *refElément* la référence de l'élément racine (créé par exemple à l'aide de la commande **DOM Creer ref XML**).

Passez dans *xChemin* le chemin d'accès de l'élément à créer en notation XPath (cf. paragraphe "Utilisation de la notation XPath" dans la section **Présentation des commandes XML DOM**). Si des éléments du chemin n'existent pas, ils sont créés.
Il est possible de passer directement dans *xChemin* un nom d'élément simple afin de créer un sous-élément à partir de l'élément courant (cf. exemple 3).

Note : Si vous avez défini un ou plusieurs espace(s) de nommage pour l'arbre désigné par *refElément* (cf. commande **DOM Creer ref XML**), vous devez préfixer le paramètre *xChemin* du nom de l'espace à utiliser (par exemple "MonNameSpace:MonElément").

Vous pouvez passer dans les paramètres facultatifs *nomAttribut* et *valeurAttribut* un couple attribut / valeur d'attribut (sous forme de variables, champs ou valeurs littérales). Vous pouvez passer autant de couples que vous voulez.
Le paramètre *valeurAttribut* peut être de type texte ou d'un autre type (booléen, entier, réel, heure ou date). Si vous passez une valeur d'un type autre que texte, 4D se charge de la conversion en texte, selon les principes suivants :

Type	Exemple de valeur convertie
Booléen	"true" ou "false" (non traduit)
Entier	"123456"
Réel	"12.34" (le séparateur décimal est toujours ".")
Heure	"5233" (nombre de secondes)
Date	"2006-12-04T00:00:00Z" (norme RFC 3339)

La commande retourne en résultat la référence XML de l'élément créé.

Exemple 1

Nous souhaitons créer l'élément suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <RootElement>    <Elem1>       <Elem2>          <Elem3>
</Elem3>          <Elem3> </Elem3>       </Elem2>    </Elem1> </RootElement>
```

Pour cela, il suffit d'écrire :

```
C_TEXTE(vRefRacine;vRefElement)
vRefRacine:=DOM Creer ref XML("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElement:=DOM Creer element XML(vRefRacine;vxPath)
vxPath:="/RootElement/Elem1/Elem2/Elem3[2]"
vRefElement:=DOM Creer element XML(vRefRacine;vxPath)
```

Exemple 2

Nous souhaitons créer l'élément suivant (comportant des attributs) :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <RootElement>    <Elem1>       <Elem2>          <Elem3
Font=Verdana Size=10> </Elem3>       </Elem2>    </Elem1> </RootElement>
```

Pour cela, il suffit d'écrire :

```
C_TEXTE(vRefRacine;vRefElement)
C_TEXTE($aAttrNom1;$aAttrNom2;$aAttrVal1;$aAttrVal2)
$aAttrNom1:="Font"
$aAttrNom2:="Size"
$aAttrVal1:="Verdana"
$aAttrVal2:="10"

vRefRacine:=DOM Creer ref XML("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElement:=DOM Creer element XML(vRefRacine;vxPath;$aAttrNom1;$aAttrVal1;$aAttrNom2;$aAttrVal2)
```

Exemple 3

Nous souhaitons créer et exporter la structure suivante :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <Racine>    <Elem1>Hello</Elem1> </Racine>
```

Nous souhaitons utiliser la syntaxe basée sur un nom d'élément simple. Pour cela, il suffit d'écrire :

```
C_TEXTE($root)
C_TEXTE($ref1)

$root:=DOM Creer ref XML("Racine")
$ref1:=DOM Creer element XML($root;"Elem1")
DOM ECRIRE VALEUR ELEMENT XML($ref1;"Hello")
DOM EXPORTER VERS FICHIER($root;"mondoc.xml")
DOM FERMER XML($root)
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

Une erreur est générée lorsque :

- la référence de l'élément racine n'est pas valide
- le nom de l'élément à créer n'est pas valide (par exemple, s'il débute par un chiffre).

DOM Creer element XML tableaux

DOM Creer element XML tableaux (refElément ; xChemin {; tabNomsAttributs ; tabValeursAttributs} {; tabNomsAttributs2 ; tabValeursAttributs2 ; ... ; tabNomsAttributsN ; tabValeursAttributsN}) -> Résultat

Paramètre	Type		Description
refElément	Texte	➡	Référence d'élément XML racine
xChemin	Texte	➡	Chemin XPath de l'élément XML à créer
tabNomsAttributs	Tableau chaîne	➡	Tableau de noms d'attributs
tabValeursAttributs	Tableau chaîne	➡	Tableau de valeurs d'attributs
Résultat	Texte	⬅	Référence de l'élément XML créé

Description

La commande **DOM Creer element XML tableaux** permet d'ajouter un nouvel élément dans l'élément XML *refElément* ainsi que, facultativement, des attributs et leurs valeurs sous forme de tableaux.

Hormis la prise en charge de tableaux (cf. ci-dessous), cette commande est identique à **DOM Creer element XML**. Reportez-vous à la description de cette commande pour le détail de son fonctionnement.

Facultativement, la commande **DOM Creer element XML tableaux** permet de passer plusieurs couples d'attributs et de valeurs d'attributs sous forme de tableaux dans les paramètres *tabNomsAttributs* et *tabValeursAttributs*. Vous pouvez passer dans *tabValeursAttributs* des tableaux de type texte, date, numérique et image. 4D effectue automatiquement les conversions nécessaires, vous pouvez modifier ces conversions à l'aide de la commande **XML FIXER OPTIONS**.

Les tableaux doivent avoir été créés au préalable et fonctionner par paires. Vous pouvez passer autant de couples de tableaux et autant d'éléments dans chaque couple que vous voulez.

Exemple

Nous souhaitons créer l'élément suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <RootElement>    <Elem1>       <Elem2>          <Elem3
Font="Verdana" Size="10" Style="Bold"></Elem3>       </Elem2>    </Elem1> </RootElement>
```

Pour cela, il suffit d'écrire :

```
TABLEAU TEXTE(tNomsAtt;3)
TABLEAU TEXTE(tValeursAtt;3)
tNomsAtt{1}:="Font"
tValeursAtt{1}:="Verdana"
tNomsAtt{2}:="Size"
tValeursAtt{2}:="10"
tNomsAtt{3}:="Style"
tValeursAtt{3}:="Bold"
vRefRacine:=DOM Creer ref XML("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElement:=DOM Creer element XML tableaux(vRefRacine;vxPath;tNomsAtt;tValeursAtt)
```

DOM Creer ref XML

DOM Creer ref XML (racine {; nameSpace} {; nSNom ; nSValeur} {; nSNom2 ; nSValeur2 ; ... ; nSNomN ; nSValeurN}) -> Résultat

Paramètre	Type		Description
racine	Chaîne	➡	Nom de l'élément racine
nameSpace	Chaîne	➡	Valeur de l'espace de nommage (Namespace)
nSNom	Chaîne	➡	Nom d'espace de nommage
nSValeur	Chaîne	➡	Valeur d'espace de nommage
Résultat	Chaîne	⬅	Référence de l'élément XML racine

Description

La commande **DOM Creer ref XML** crée un arbre XML vide en mémoire et retourne sa référence.

Passez dans le paramètre *racine* le nom de l'élément racine de l'arbre XML.

Passez dans le paramètre facultatif *nameSpace* la déclaration de la valeur de l'espace de nommage (namespace) de l'arbre (par exemple "http://www.4d.com").
A noter qu'il est possible de préfixer le paramètre *racine* avec le nom de l'espace de nommage, suivi de *:* (par exemple "MonNameSpace:MaRacine"). Dans ce cas, le paramètre *nameSpace* précisant la valeur de l'espace de nommage est obligatoire.

Note : L'espace de nommage (namespace) est une chaîne de caractères permettant de garantir l'unicité des noms de variables XML. En général, un URL du type http://www.monsite.com/monurl est utilisé. Il n'est pas nécessaire que l'URL soit valide sur le site, il faut juste qu'il soit unique.

Vous pouvez déclarer un ou plusieurs espace(s) de nommage supplémentaire(s) dans l'arbre XML généré, à l'aide de couples *nSNom / nSValeur*. Vous pouvez passer autant de couples nom / valeur d'espace de nommage que vous voulez.

Important : N'oubliez pas d'appeler la commande **DOM FERMER XML** afin de libérer la mémoire lorsque vous avez terminé d'utiliser l'arbre XML.

Exemple 1

Création d'un arbre XML simple :

```
C_TEXTE(vRefElem)
vRefElem:=DOM Creer ref XML("MaRacine")
```

Ce code produit le résultat suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <MaRacine/>
```

Exemple 2

Création d'un arbre XML avec un espace de nommage :

```
C_TEXTE(vRefElem)
$Racine:="MonNameSpace:MaRacine"
$Namespace:="http://www.4D.com/tech/namespace"
vRefElem:=DOM Creer ref XML($Racine;$Namespace)
```

Ce code produit le résultat suivant :

```
<MonNameSpace:MaRacine xmlns:MonNameSpace="http://www.4D.com/tech/namespace"/>
```

Exemple 3

Création d'un arbre XML avec plusieurs espaces de nommage :

```
C_TEXTE(vRefElem)
C_TEXTE($aNSNom1;$aNSNom2;$aNSValeur1;$aNSValeur2)
$Racine:="MonNameSpace:MaRacine"
$Namespace:="http://www.4D.com/tech/namespace"
$aNSNom1:="NSNom1"
$aNSNom2:="NSNom2"
$aNSValeur1:="http://www.4D.com/Prod/namespace"
$aNSValeur2:="http://www.4D.com/Mkt/namespace"
vRefElem:=DOM Creer ref XML($Racine;$Namespace;$aNSNom1;$aNSValeur1;$aNSNom2;$aNSValeur2)
```

Ce code produit le résultat suivant :

```
<MonNameSpace:MaRacine xmlns:MonNameSpace="http://www.4D.com/tech/nameSpace"
NSNom1="http://www.4D.com/Prod/namespace" NSNom2="http://www.4D.com/Mkt/namespace"/>
```

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

DOM ECRIRE ATTRIBUT XML

DOM ECRIRE ATTRIBUT XML (refElément ; nomAttribut ; valeurAttribut {; nomAttribut2 ; valeurAttribut2 ; ... ; nomAttributN ; valeurAttributN})

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML
nomAttribut	Chaîne	➡	Attribut à définir
valeurAttribut	Chaîne, Booléen, Entier long, Réel, Heure, Date	➡	Nouvelle valeur d'attribut

Description

La commande **DOM ECRIRE ATTRIBUT XML** permet d'ajouter un ou plusieurs attribut(s) à l'élément XML dont la référence est passée dans le paramètre *refElément*. Elle permet également de définir la valeur de chaque attribut défini.

Passez dans les paramètres *nomAttribut* et *valeurAttribut* respectivement l'attribut à écrire et sa valeur (sous forme de variables, champs ou valeurs littérales). Vous pouvez passer autant de couples attribut/valeur que vous voulez.
Le paramètre *valeurAttribut* peut être de type texte ou d'un autre type (booléen, entier, réel, heure ou date). Si vous passez une valeur d'un type autre que texte, 4D se charge de la conversion en texte, selon les principes suivants :

Type	Exemple de valeur convertie
Booléen	"true" ou "false" (non traduit)
Entier	"123456"
Réel	"12.34" (le séparateur décimal est toujours ".")
Heure	"5233" (nombre de secondes)
Date	"2006-12-04T00:00:00Z" (norme RFC 3339)

Exemple

Soit la source XML suivante :

```
<Book>    <Title>The Best Seller</Title> </Book>
```

Si le code suivant est exécuté :

```
vAttrName:="Font"
vAttrVal:="Verdana"
DOM ECRIRE ATTRIBUT XML(vRefElem;vAttrName;vAttrVal)
```

Nous obtenons :

```
<Book>    <Title Font=Verdana>The Best Seller</Title> </Book>
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

DOM ECRIRE NOM ELEMENT XML

DOM ECRIRE NOM ELEMENT XML (refElément ; nomElément)

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML
nomElément	Chaîne	➡	Nouveau nom de l'élément

Description

La commande **DOM ECRIRE NOM ELEMENT XML** permet de modifier le nom de l'élément désigné par *refElément*.

Passez dans *refElément* la référence de l'élément à renommer et dans *nomElément* le nouveau nom de l'élément. Bien entendu, la commande se charge de modifier les balises d'ouverture et de fermeture de l'élément.

Exemple

Soit la source XML suivante :

```
<Book>    <Title>The Best Seller</Title> </Book>
```

Si le code suivant est exécuté, en admettant que *vRefElem* contienne la référence de l'élément 'Book' :

```
DOM ECRIRE NOM ELEMENT XML(vRefElem;"BestSeller")
```

Nous obtenons :

```
<BestSeller>    <Title>The Best Seller</Title> </BestSeller>
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

Une erreur est générée lorsque :

- la référence de l'élément n'est pas valide
- le nouveau nom d'élément n'est pas valide (par exemple, s'il débute par un chiffre).

DOM ECRIRE VALEUR ELEMENT XML

DOM ECRIRE VALEUR ELEMENT XML (refElément {; xChemin}; valeurElément {; *})

Paramètre	Type		Description
refElément	Chaîne	⇒	Référence d'élément XML
xChemin	Texte	⇒	Chemin XPath de l'élément XML
valeurElément	Chaîne, Variable	⇒	Nouvelle valeur de l'élément
*	Opérateur	⇒	Si passé : définir la valeur en CDATA

Description

La commande **DOM ECRIRE VALEUR ELEMENT XML** permet de modifier la valeur de l'élément désigné par *refElément*.

Si vous passez le paramètre facultatif *xChemin*, vous choisissez d'utiliser la notation XPath pour désigner l'élément à modifier (pour plus d'informations sur cette notation, reportez-vous au paragraphe "Utilisation de la notation XPath" dans la section **Présentation des commandes XML DOM**). Dans ce cas, vous devez passer la référence d'un élément XML racine dans *refElément* et le chemin XPath de l'élément à modifier dans *xChemin*.

Passez dans *valeurElément* une chaîne ou une variable (ou un champ) contenant la nouvelle valeur de l'élément :

- si vous passez une chaîne, la valeur sera affectée telle quelle dans la structure XML.
- si vous passez une variable ou un champ, 4D appliquera un traitement approprié à la valeur en fonction du type de *valeurElément*. Tous les types de données peuvent être utilisés, à l'exception des tableaux, images et pointeurs.

Lorsque le paramètre facultatif astérisque (*) est passé, vous indiquez que la valeur de l'élément doit être définie sous la forme CDATA. La forme spéciale CDATA permet d'écrire du texte sous forme brute (cf. exemple 2).

Note : Lorsque l'élément désigné par *refElément* est de type BLOB, **DOM ECRIRE VALEUR ELEMENT XML** l'encode automatiquement en base64. Dans ce cas, la commande **DOM LIRE VALEUR ELEMENT XML** effectue automatiquement l'opération inverse.

Exemple 1

Soit la source XML suivante :

```
<Book>    <Title>The Best Seller</Title> </Book>
```

Si le code suivant est exécuté, en admettant que *vRefElem* contienne la référence de l'élément 'Title' :

```
DOM ECRIRE VALEUR ELEMENT XML(vRefElem;"The Loser")
```

Nous obtenons :

```
<Book>    <Title>The Loser</Title> </Book>
```

Exemple 2

Soit la source XML suivante :

```
<Maths>    <Postulate>1+2=3</Postulate> </Maths>
```

Nous souhaitons écrire le texte "12 < 18" dans l'élément *<postulate>*. Cette chaîne ne peut pas être écrite telle quelle en XML car le caractère "<" n'est pas accepté. Ce caractère doit donc être transformé en "<", ou la forme CDATA doit être utilisée. Si *vElemRef* désigne le noeud XML *<Postulate>* :

```
` Forme normale
DOM ECRIRE VALEUR ELEMENT XML(vRefElem;"12 < 18")
```

Nous obtenons :

```
<Maths>    <Postulate>12 < 18</Postulate> </Maths>
```

```
` Forme CDATA
DOM ECRIRE VALEUR ELEMENT XML(vRefElem;"12 < 18";*)
```

Nous obtenons :

```
<Maths>    <Postulate><![CDATA[12 < 18]]></Postulate> </Maths>
```

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée (par exemple, si la référence de l'élément n'est pas valide).

DOM EXPORTER VERS FICHIER

DOM EXPORTER VERS FICHIER (refElément ; cheminFichier)

Paramètre	Type		Description
refElément	Chaîne	⇨	Référence d'élément XML racine
cheminFichier	Texte	⇨	Chemin d'accès complet du fichier

Description

La commande **DOM EXPORTER VERS FICHIER** permet de sauvegarder un arbre XML dans un fichier sur disque.

Passez dans *refElément* la référence de l'élément racine à exporter.
Passez dans *cheminFichier* le chemin d'accès complet du fichier d'export à utiliser ou à créer. Si le fichier n'existe pas, il est créé.
Si vous passez uniquement un nom de fichier (sans chemin d'accès), le fichier sera recherché ou créé à côté du fichier de structure.
Si vous passez une chaîne vide (""), une boîte de dialogue standard d'ouverture et de création de fichier apparaît.

Notes sur le traitement des caractères de fin de ligne

En XML, les retours à la ligne ne sont pas significatifs, qu'ils soient présents à l'intérieur ou entre des éléments XML. En interne, le XML utilise des caractères normalisés LF comme séparateurs de lignes.

Lors des opérations d'importation et d'exportation, les caractères de retour à la ligne peuvent donc être convertis. A l'importation, l'analyseur XML remplace les caractères CRLF (retours à la ligne standard sous Windows) par des caractères LF. A l'export, les LF sont remplacés par des caractères CR.

Si vous souhaitez préserver des retours chariot, il faut les inclure dans un élément XML CDATA pour qu'ils ne soient pas traités par l'analyseur XML. Vous pouvez également utiliser le caractère "
" qui est un retour chariot explicite et qui ne sera pas traité par l'analyseur en lieu et place des caractères CRLF.

Exemple

Cet exemple sauvegarde l'arbre *vRefElem* dans le fichier MonDoc.xml :

```
DOM EXPORTER VERS FICHIER(vRefElem;"C:\\dossier\MonDoc.xml")
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

Une erreur est générée lorsque :

- la référence de l'élément n'est pas valide,
- le chemin d'accès spécifié n'est pas valide,
- le volume de stockage retourne une erreur (disque plein, etc.).

DOM EXPORTER VERS VARIABLE

DOM EXPORTER VERS VARIABLE (refElément ; vVarXml)

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML racine
vVarXml	Texte, BLOB	➡	Variable devant recevoir l'arbre XML

Description

La commande **DOM EXPORTER VERS VARIABLE** permet de sauvegarder un arbre XML dans une variable texte ou BLOB.

Passez dans *refElément* la référence de l'élément racine à exporter.

Passez dans *vVarXml* le nom de la variable devant contenir l'arbre XML. Cette variable peut être de type Texte ou BLOB. Vous pouvez choisir le type en fonction des opérations à effectuer par la suite ou de la taille que l'arbre peut atteindre (rappelons qu'en mode Non unicode les variables de type Texte sont limitées à 32 K de texte, en mode Unicode cette limite est de 2 Go).

Attention, en mode Non unicode si vous utilisez une variable Texte pour stocker l'élément *refElément*, il sera encodé en Mac "courant" (c'est-à-dire Mac Roman sur la plupart des systèmes occidentaux). Cela signifie que le texte retourné ne sera plus dans l'encodage d'origine (encoding="xxx"). Dans ce cas, la variable *vVarXml* permet de visualiser ou de stocker le code obtenu mais PAS de générer un document XML valide (via la commande **ENVOYER PAQUET** par exemple).

En mode Unicode, l'encodage d'origine est conservé dans la variable.

Notes sur le traitement des caractères de fin de ligne

En XML, les retours à la ligne ne sont pas significatifs, qu'ils soient présents à l'intérieur ou entre des éléments XML. En interne, le XML utilise des caractères normalisés LF comme séparateurs de lignes.

Lors des opérations d'importation et d'exportation, les caractères de retour à la ligne peuvent donc être convertis. A l'importation, l'analyseur XML remplace les caractères CRLF (retours à la ligne standard sous Windows) par des caractères LF. A l'export, les LF sont remplacés par des caractères CR.

Si vous souhaitez préserver des retours chariot, il faut les inclure dans un élément XML CDATA pour qu'ils ne soient pas traités par l'analyseur XML. Vous pouvez également utiliser le caractère "
" qui est un retour chariot explicite et qui ne sera pas traité par l'analyseur en lieu et place des caractères CRLF.

Exemple

Cet exemple sauvegarde l'arbre vRefElem dans une variable texte :

```
C_TEXTE(vtMonTexte)
DOM EXPORTER VERS VARIABLE(vRefElem;vtMonTexte)
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée (par exemple, si la référence de l'élément n'est pas valide).

DOM FERMER XML

DOM FERMER XML (refElément)

Paramètre	Type		Description
refElément	Chaîne	⇨	Référence d'élément XML racine

Description

La commande **DOM FERMER XML** libère l'espace mémoire occupé par l'objet XML désigné par *refElément*.
Si *refElément* n'est pas un objet XML racine, une erreur est générée.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM FIXER DECLARATION XML

DOM FIXER DECLARATION XML (refElément ; encodage {; autonome {; indentation}})

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML
encodage	Chaîne	➡	Jeu de caractères du document XML
autonome	Booléen	➡	Vrai=le document est autonome Faux (défaut)=le document n'est pas autonome
indentation	Booléen	➡	*** Obsolète, ne plus utiliser ***

Description

La commande **DOM FIXER DECLARATION XML** permet de définir diverses options qui seront utilisée pour la création de l'arbre XML désigné par *refElément*. Ces options concernent l'encodage et l'attribut autonome (standalone) de l'arbre :

- *encodage* : indique le jeu de caractères employé. Par défaut (si la commande n'est pas appelée), le jeu de caractères UTF-8 (Unicode compressé) est utilisé.
  Note : Si vous passez un jeu de caractères non pris en charge par les commandes XML de 4D, l'UTF-8 sera utilisé. Reportez-vous au paragraphe **Jeux de caractères** pour connaître la liste des jeux de caractères pris en charge (l'UTF-8 est toutefois recommandé dans la plupart des cas).
- *autonome* : indique si l'arbre est autonome (**Vrai**) ou s'il dépend, pour son fonctionnement, de ressources externes (**Faux**). Par défaut (si la commande n'est pas appelée ou si le paramètre est omis), l'arbre n'est pas autonome.

Note de compatibilité : Le paramètre *indentation* est conservé pour des raisons de compatibilité avec les versions précédentes de 4D mais son usage est déconseillé à compter de 4D v12. Désormais, pour définir l'indentation du document, il est recommandé d'utiliser la commande **XML FIXER OPTIONS**.

Exemple

Cet exemple définit l'encodage et l'option standalone de l'élément *refElément* :

```
DOM FIXER DECLARATION XML(refElément;"UTF-16";Vrai)
```

DOM Inserer element XML

DOM Inserer element XML (refElémentCible ; refElémentSource ; indexEnfant) -> Résultat

Paramètre	Type		Description
refElémentCible	Texte	⇒	Référence de l'élément XML parent
refElémentSource	Texte	⇒	Référence de l'élément XML à insérer
indexEnfant	Entier long	⇒	Index de l'enfant de l'élément cible avant lequel le nouvel élément doit être inséré
Résultat	Texte	⇐	Référence du nouvel élément XML

Description

La commande **DOM Inserer element XML** permet d'insérer un nouvel élément XML parmi les enfants de l'élément XML dont la référence est passée dans le paramètre *refElémentCible*.

Passez dans *refElémentSource* l'élément à insérer. Cet élément doit être passé en tant que référence d'un élément XML existant dans un arbre DOM.

Le paramètre *indexEnfant* permet de désigner l'enfant de l'élément parent avant lequel le nouvel élément doit être inséré. Passez un numéro d'index dans ce paramètre. Si la valeur est invalide (par exemple s'il n'existe pas d'élément enfant de cet index), le nouvel élément sera ajouté avant le premier enfant de l'élément parent.

La commande retourne la référence de l'élément XML obtenu.

Exemple

Dans la structure suivante, nous souhaitons inverser le premier et le deuxième livre :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <BookCatalog>   <Book>           <Title>Services Web Open
Source</Title>           <Author>Collectif</Author>           <Date>2003</Date>           <ISBN>2-7440-1507-
5</ISBN>           <Publisher>Wrox</Publisher>     </Book> <Book>           <Title>Construire des services Web
XML</Title>           <Author>Scott Short</Author>           <Date>2002</Date>           <ISBN>2-10-006476-
2</ISBN>           <Publisher>Microsoft Press</Publisher>     </Book> </BookCatalog>
```

Pour cela, il suffit d'exécuter le code suivant :

```
C_TEXTE($refRoot)
$refRoot:=DOM Analyser source XML("") //sélection du document XML
Si(OK=1)
   C_TEXTE($newStruct)
   $newStruct:=DOM Creer ref XML("BookCatalog")

   $refBook:=DOM Chercher element XML($refRoot;"/BookCatalog/Book[1]")
   $refNewElement:=DOM Ajouter element XML($newStruct;$refBook)

   $refBook:=DOM Chercher element XML($refRoot;"/BookCatalog/Book[2]")
   C_TEXTE($refNewElement)
   $refNewElement:=DOM Inserer element XML($newStruct;$refBook;1)

   DOM FERMER XML($newStruct)
   DOM FERMER XML($refRoot)
Fin de si
```

DOM LIRE ATTRIBUT XML PAR INDEX

DOM LIRE ATTRIBUT XML PAR INDEX (refElément ; indexAttribut ; nomAttribut ; valeurAttribut)

Paramètre	Type		Description
refElément	Chaîne	⇨	Référence d'élément XML
indexAttribut	Entier long	⇨	Numéro d'indice de l'attribut
nomAttribut	Variable	⇦	Nom de l'attribut
valeurAttribut	Variable	⇦	Valeur de l'attribut

Description

La commande **DOM LIRE ATTRIBUT XML PAR INDEX** permet de connaître le nom ainsi que la valeur d'un attribut désigné par son numéro d'indice.

Passez dans *refElément* la référence d'un élément XML et dans *indexAttribut* le numéro d'indice de l'attribut dont vous voulez connaître le nom. Le nom est retourné dans le paramètre *nomAttribut* et sa valeur est retournée dans le paramètre *valeurAttribut*. 4D tentera de convertir la valeur obtenue dans le type de la variable passée en paramètre.

Note : Le numéro d'indice ne correspond pas à l'emplacement de l'attribut dans le fichier XML affiché sous forme de texte. En XML, l'indice d'un attribut indique sa position parmi les attributs classés par ordre alphabétique (en fonction de leur nom). Pour une illustration de ce principe, reportez-vous à l'exemple de la commande **DOM Compter attributs XML**.

Si la valeur passée dans *indexAttribut* est supérieure au nombre d'attributs présents dans l'élément XML, une erreur est retournée.

Exemple

Reportez-vous à l'exemple de la commande **DOM Compter attributs XML**.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM LIRE ATTRIBUT XML PAR NOM

DOM LIRE ATTRIBUT XML PAR NOM (refElément ; nomAttribut ; valeurAttribut)

Paramètre	Type		Description
refElément	Chaîne	⇒	Référence d'élément XML
nomAttribut	Chaîne	⇒	Nom d'attribut
valeurAttribut	Variable	⇐	Valeur de l'attribut

Description

La commande **DOM LIRE ATTRIBUT XML PAR NOM** permet de connaître la valeur d'un attribut désigné par son nom.

Passez dans *refElément* la référence d'un élément XML et dans *nomAttribut* le nom d'attribut dont vous voulez connaître la valeur. La valeur est retournée dans le paramètre *valeurAttribut*.
4D tentera de convertir la valeur obtenue dans le type de la variable passée en paramètre.

Si aucun attribut *nomAttribut* n'existe dans l'élément XML, une erreur est retournée. Si plusieurs attributs de l'élément XML portent le nom spécifié, seule la valeur du premier attribut est retournée.

Exemple

Cette méthode permet de récupérer une valeur d'attribut XML à l'aide de son nom :

```
C_BLOB(maVarBlob)
C_TEXTE($ref_XML_Parent;$ref_XML_Enfant)
C_ENTIER LONG($NumLigne)

$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire premier element XML enfant($ref_XML_Parent)
DOM LIRE ATTRIBUT XML PAR NOM($ref_XML_Enfant;"N";$NumLigne)
```

Si cette méthode est appliquée à l'exemple ci-dessous, $NumLigne contient la valeur 1 :

```
<?xml version="1.0" ?>
- <STANZA>
    <LINE N="1">I heard a thousand blended notes,</LINE>
    <LINE N="2">While in grove I sate reclined,</LINE>
    <LINE N="3">In that sweet mood when pleasant thoughts</LINE>
    <LINE N="4">Bring sad thoughts to the mind.</LINE>
  </STANZA>
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM Lire dernier element XML enfant

DOM Lire dernier element XML enfant (refElément {; nomElémentEnf {; valeurElémentEnf}}) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML
nomElémentEnf	Chaîne	⬅	Nom de l'élément enfant
valeurElémentEnf	Chaîne	⬅	Valeur de l'élément enfant
Résultat	Chaîne	➲	Référence de l'élément XML (16 caractères)

Description

La commande **DOM Lire dernier element XML enfant** retourne une référence XML vers le dernier "enfant" de l'élément XML passé en référence dans *refElément*. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres *nomElémentEnf* et *valeurElémentEnf*, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément enfant.

Exemple

Récupération de la référence du dernier élément XML du parent racine. La structure XML (C:\\import.xml) est préalablement chargée dans un BLOB :

```
C_BLOB(maVarBlob)
C_TEXTE($ref_XML_Parent;$ref_XML_Enfant)
C_TEXTE($nomEnfant;$valeurEnfant)

DOCUMENT VERS BLOB("c:\\import.xml";maVarBlob)
$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire dernier element XML enfant($ref_XML_Parent;$nomEnfant;$valeurEnfant)
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

DOM Lire element XML

DOM Lire element XML (refElément ; nomElément ; indice ; valeurElément) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	⇒	Référence d'élément XML
nomElément	Chaîne	⇒	Nom de l'élément à lire
indice	Entier long	⇒	Numéro d'indice de l'élément à lire
valeurElément	Variable	⇐	Valeur de l'élément
Résultat	Chaîne	⇐	Référence de l'élément XML (16 caractères)

Description

La commande **DOM Lire element XML** retourne une référence XML vers l'élément "enfant" dépendant des paramètres *nomElément* et *index*.
La valeur de l'élément est également retournée dans le paramètre *valeurElément*.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM Lire element XML frere precedent

DOM Lire element XML frere precedent (refElément {; nomElémentFrère {; valeurElémentFrère}}) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	⇒	Référence d'élément XML
nomElémentFrère	Chaîne	⇐	Nom de l'élément XML frère
valeurElémentFrère	Chaîne	⇐	Valeur de l'élément XML frère
Résultat	Chaîne	⇒	Référence de l'élément XML frère (16 caractères)

Description

La commande **DOM Lire element XML frere precedent** retourne une référence vers le précédent "frère" de l'élément XML passé en référence. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres *nomElémentFrère* et *valeurElémentFrère*, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément "frère" précédent.

Cette commande permet de naviguer parmi les "enfants" d'un élément XML.

Avant le premier "frère", la variable système OK prend la valeur 0.

Variables et ensembles système

Si la commande a été correctement exécutée et si l'élément référencé n'est pas le premier "enfant" de la structure, la variable système OK prend la valeur 1. Si une erreur se produit ou si l'élément analysé est le premier "enfant" de la structure, elle prend la valeur 0.

DOM Lire element XML frere suivant

DOM Lire element XML frere suivant (refElément {; nomElémentFrère {; valeurElémentFrère}}) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	⇒	Référence d'élément XML
nomElémentFrère	Chaîne	⇐	Nom de l'élément XML frère
valeurElémentFrère	Chaîne	⇐	Valeur de l'élément XML frère
Résultat	Chaîne	⇒	Référence de l'élément XML frère (16 caractères)

Description

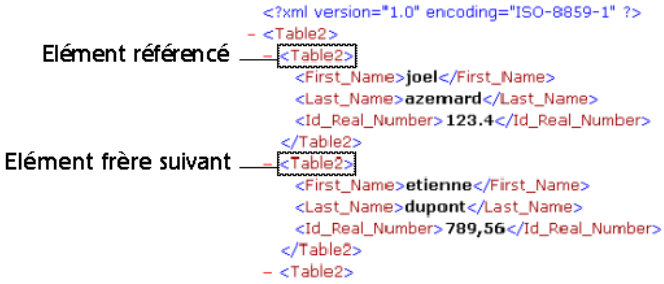
La commande **DOM Lire element XML frere suivant** retourne une référence vers le prochain "frère" de l'élément XML passé en référence. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres *nomElémentFrère* et *valeurElémentFrère*, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément "frère".
Cette commande permet de naviguer parmi les "enfants" d'un élément XML.
Après le dernier "frère", la variable système OK prend la valeur 0.

Exemple 1

Récupération de la référence de l'élément XML frère suivant l'élément passé en paramètre :

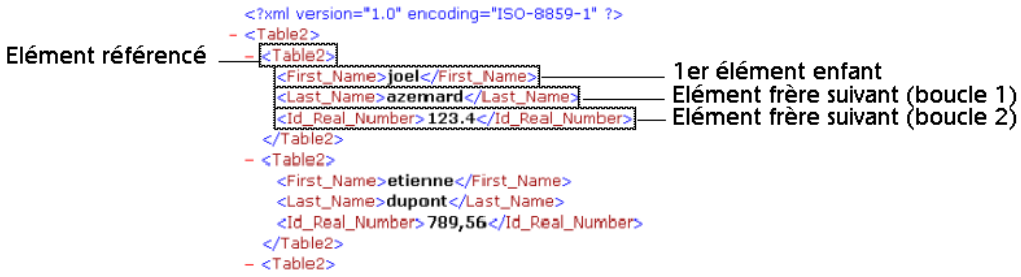
```
C_TEXTE($ref_XML_Parent;$ref_XML_Suivant)
$ref_XML_Suivant:=DOM Lire element XML frere suivant($ref_XML_Parent)
```



Exemple 2

Récupération dans une boucle des références de tous les éléments XML enfants de l'élément parent passé en paramètre, à compter du premier enfant :

```
C_TEXTE($ref_XML_Parent;$ref_XML_Premier;$ref_XML_Suivant)

$ref_XML_Premier:=DOM Lire premier element XML enfant($ref_XML_Parent)
$ref_XML_Suivant:=$ref_XML_Premier
Tant que(OK=1)
   $ref_XML_Suivant:=DOM Lire element XML frere suivant($ref_XML_Suivant)
Fin tant que
```



Variables et ensembles système

Si la commande a été correctement exécutée et si l'élément analysé n'est pas le dernier "frère" de l'élément référencé, la variable système OK prend la valeur 1. Si une erreur se produit ou si l'élément analysé est le dernier "frère" de l'élément référencé, elle prend la valeur 0.

DOM Lire element XML parent

DOM Lire element XML parent (refElément {; nomElémentPar {; valeurElémentPar}}) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	⇨	Référence d'élément XML
nomElémentPar	Chaîne	⇦	Nom de l'élément XML parent
valeurElémentPar	Chaîne	⇦	Valeur de l'élément XML parent
Résultat	Chaîne	⇨	Référence de l'élément XML parent (16 caractères)

Description

La commande **DOM Lire element XML parent** retourne une référence XML vers le "parent" de l'élément XML passé en référence dans *refElément*. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres *nomElémentPar* et *valeurElémentPar*, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément parent.

Si vous utilisez cette commande en passant un élément racine dans *refElément*, la commande retourne la référence "#document". Le noeud document est le parent d'un élément racine.
Si vous utilisez cette commande sur un noeud document, la commande retourne une référence nulle ("0000000000000000") et la variable OK prend la valeur 0.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

DOM Lire element XML racine

DOM Lire element XML racine (refElément) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	⇨	Référence d'élément XML
Résultat	Chaîne	⇦	Référence de l'élément racine (16 caractères) ou "" en cas d'erreur

Description

La commande **DOM Lire element XML racine** retourne une référence vers l'élément racine du document auquel appartient l'élément XML passé dans le paramètre *refElément*. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

DOM Lire informations XML

DOM Lire informations XML (refElément ; infoXML) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	⇒	Référence d'élément XML racine
infoXML	Entier long	⇒	Type d'information à lire
Résultat	Chaîne	⇐	Valeur de l'information XML

Description

La commande **DOM Lire informations XML** permet de récupérer diverses informations sur l'élément XML désigné par *refElément*.

Passez dans *infoXML* un code indiquant le type d'information à récupérer. Vous pouvez utiliser les constantes prédéfinies suivantes, placées dans le thème "**XML**" :

Constante	Type	Valeur	Comment
Encoding	Entier long	4	Encodage utilisé (UTF-8, ISO...)
ID PUBLIC	Entier long	1	Identificateur public (FPI) de la DTD à laquelle le document se conforme (si la balise DOCTYPE xxx PUBLIC est présente)
ID SYSTEM	Entier long	2	Identificateur système
Nom DOCTYPE	Entier long	3	Nom de l'élément racine tel que défini dans la balise DOCTYPE
URI document	Entier long	6	URI de la DTD
Version	Entier long	5	Version de XML accepté

DOM LIRE NOEUDS ENFANTS XML

DOM LIRE NOEUDS ENFANTS XML (refElément ; tabTypesEnfants ; tabRefsNoeuds)

Paramètre	Type		Description
refElément	Texte	⇒	Référence d'élément XML
tabTypesEnfants	Tableau entier long	⇐	Types des noeuds enfants
tabRefsNoeuds	Tableau texte	⇐	Références ou Valeurs des noeuds enfants

Description

La commande **DOM LIRE NOEUDS ENFANTS XML** retourne les types et les références ou valeurs de tous les noeuds enfants de l'élément XML désigné par *refElément*.

Les types des noeuds enfants sont retournés dans le tableau *tabTypesEnfants*. Vous pouvez comparer les valeurs renvoyées par la commande avec les constantes suivantes, placées dans le thème **XML** :

Constante	Type	Valeur
XML commentaire	Entier long	2
XML instruction de traitement	Entier long	3
XML Donnée	Entier long	6
XML CDATA	Entier long	7
XML DOCTYPE	Entier long	10
XML Elément	Entier long	11

Pour plus d'informations, reportez-vous à la description de la commande **DOM Ajouter noeud enfant XML**.

Le tableau *tabRefsNoeuds* reçoit les valeurs ou les références des éléments en fonction de leur nature (contenus ou instructions).

Exemple

Soit la structure XML suivante :

```
<monElement>Bonjour<br/>La<br/>FRANCE</monElement>
```

Après l'exécution de ces instructions :

```
refElement:=DOM Chercher element XML($root;"monElement")
DOM LIRE NOEUDS ENFANTS XML(refElement;$tabtype;$tabtext)
```

... les tableaux $tabtype et $tabtext contiendront les valeurs suivantes :

$tabtype{1}=6	$tabtext{1} = "Bonjour"
$tabtype{2}=11	$tabtext{2} = "AEF1233456878977" (référence de l'élément)
$tabtype{3}=6	$tabtext{3} = "La"
$tabtype{4}=11	$tabtext{4} = "AEF1237897734568" (référence de l'élément)
$tabtype{5}=6	$tabtext{5} = "FRANCE"

DOM LIRE NOM ELEMENT XML

DOM LIRE NOM ELEMENT XML (refElément ; nomElément)

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML
nomElément	Variable	⬅	Nom de l'élément

Description

La commande **DOM LIRE NOM ELEMENT XML** retourne dans le paramètre *nomElément* le nom de l'élément XML désigné par *refElément*. Pour plus d'informations sur les noms d'éléments XML, reportez-vous à la section **Présentation des commandes XML DOM**.

Exemple

Cette méthode retourne le nom de l'élément $ref_XML_Elément :

```
C_TEXTE($ref_XML_Elément)
C_TEXTE($nom)

DOM LIRE NOM ELEMENT XML($ref_XML_Elément;$nom)
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM Lire premier element XML enfant

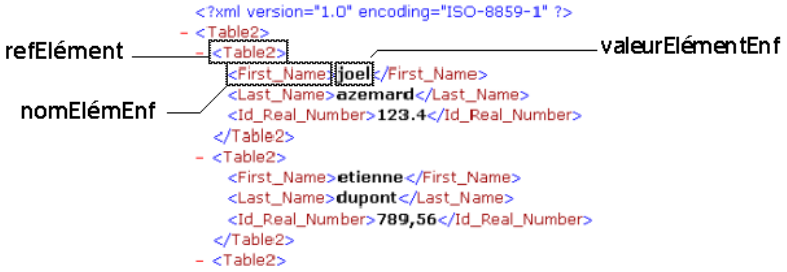
DOM Lire premier element XML enfant (refElément {; nomElémentEnf {; valeurElémentEnf}}) -> Résultat

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML
nomElémentEnf	Chaîne	⬅	Nom de l'élément XML enfant
valeurElémentEnf	Chaîne	⬅	Valeur de l'élément XML enfant
Résultat	Chaîne	➲	Référence de l'élément XML enfant (16 caractères)

Description

La commande **DOM Lire premier element XML enfant** retourne une référence XML vers le premier "enfant" de l'élément XML passé en référence dans *refElément*. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres *nomElémentEnf* et *valeurElémentEnf*, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément enfant.



Exemple 1

Récupération de la référence du premier élément XML du parent racine. La structure XML (C:\\import.xml) est préalablement chargée dans un BLOB :

```
C_BLOB(maVarBlob)
C_TEXTE($ref_XML_Parent;$ref_XML_Enfant)

DOCUMENT VERS BLOB("c:\\import.xml";maVarBlob)
$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire premier element XML enfant($ref_XML_Parent)
```

Exemple 2

Récupération de la référence, du nom et de la valeur du premier élément XML du parent racine. La structure XML (C:\\import.xml) est préalablement chargée dans un BLOB :

```
C_BLOB(maVarBlob)
C_TEXTE($ref_XML_Parent;$ref_XML_Enfant)
C_TEXTE($enfantNom;$enfantValeur)

DOCUMENT VERS BLOB("c:\\import.xml";maVarBlob)
$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire premier element XML enfant($ref_XML_Parent;$enfantNom;$enfantValeur)
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

DOM Lire ref document XML

DOM Lire ref document XML (refElément) -> Résultat

Paramètre	Type		Description
refElément	Texte	⇒	Référence d'un élément existant dans un arbre DOM
Résultat	Texte	⇐	Référence du premier élément de l'arbre DOM (noeud document)

Description

La commande **DOM Lire ref document XML** permet de récupérer la référence de l'élément "document" de l'arbre DOM dont vous avez passé la référence dans *refElément*. L'élément document est le premier élément d'un arbre DOM ; c'est le parent de l'élément racine.

La référence de l'élément document vous permet de manipuler les noeuds "Doctype" et "Instructions de traitement". Elle ne peut être utilisée qu'avec les commandes **DOM Ajouter noeud enfant XML** et **DOM LIRE NOEUDS ENFANTS XML**.

A ce niveau, vous pouvez uniquement ajouter des instructions de traitement, des commentaires ou remplacer le noeud Doctype. Vous ne pouvez pas y créer de noeud CDATA ou texte.

Exemple

Dans cet exemple nous cherchons à retrouver la déclaration de DTD du document XML :

```
C_TEXTE($refRoot)
$refRoot:=DOM Analyser source XML("")
Si(OK=1)
   C_TEXTE($refDocument)
 // on cherche le noeud document, puisque c'est le noeud auquel est
 // rattaché le nœud DOCTYPE avant le noeud root
   $refDocument:=DOM Lire ref document XML($refRoot)
   TABLEAU TEXTE($arrType;0)
   TABLEAU TEXTE($arrValue;0)
 // sur ce nœud on cherche parmi les enfants le nœud de type DOCTYPE
   DOM LIRE NOEUDS ENFANTS XML($refDocument;$arrType;$arrValue)
   C_TEXTE($text)
   $text:=""
   $pos:=Chercher dans tableau($arrType;XML DOCTYPE)
   Si($pos>-1)
 // On récupère dans $text la déclaration de DTD
      $text:=$text+"Doctype: "+$arrValue{$pos}+Caractere(Retour chariot)
   Fin de si
   DOM FERMER XML($refRoot)
Fin de si
```

DOM LIRE VALEUR ELEMENT XML

DOM LIRE VALEUR ELEMENT XML (refElément ; valeurElément {; cDATA})

Paramètre	Type		Description
refElément	Chaîne	➡	Référence d'élément XML
valeurElément	Variable	⬅	Valeur de l'élément
cDATA	Variable	⬅	Contenu de la section CDATA

Description

La commande **DOM LIRE VALEUR ELEMENT XML** retourne dans le paramètre *valeurElément* la valeur de l'élément XML désigné par *refElément*. 4D tentera de convertir la valeur obtenue dans le type de la variable passée en paramètre.

Le paramètre facultatif *cDATA* permet de récupérer le contenu de la ou des section(s) CDATA de l'élément XML *refElément* le cas échéant. Comme pour le paramètre *valeurElément*, 4D tentera de convertir la valeur obtenue dans le type de la variable passée en paramètre.

Note : Si l'élément désigné par *refElément* est un BLOB traité par la commande **DOM ECRIRE VALEUR ELEMENT XML**, il a été automatiquement encodé en base64. Par conséquent, la commande tentera automatiquement de le décoder en base64.

Exemple

Cette méthode retourne la valeur de l'élément $ref_XML_Elément :

```
C_TEXTE($ref_XML_Elément)
C_REEL($valeur)

DOM LIRE VALEUR ELEMENT XML($ref_XML_Elément;$valeur)
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM SUPPRIMER ATTRIBUT XML

DOM SUPPRIMER ATTRIBUT XML (refElément ; nomAttribut)

Paramètre	Type		Description
refElément	Texte	➡	Référence d'élément XML
nomAttribut	Texte	➡	Attribut à supprimer

Description

La commande **DOM SUPPRIMER ATTRIBUT XML** supprime, s'il existe, l'attribut désigné par *nomAttribut* de l'élément XML dont la référence est passée dans le paramètre *refElément*.

Si l'attribut a été correctement supprimé, la variable système OK prend la valeur 1. Si aucun attribut nommé *nomAttribut* n'existe dans *refElément*, une erreur est retournée et la variable système OK prend la valeur 0.

Exemple

Soit la structure suivante :

```
<?xml version="1.0" ?>
- <STANZA>
    <LINE N="1">I heard a thousand blended notes,</LINE>
    <LINE N="2">While in grove I sate reclined,</LINE>
    <LINE N="3">In that sweet mood when pleasant thoughts</LINE>
    <LINE N="4">Bring sad thoughts to the mind.</LINE>
  </STANZA>
```

Le code suivant permet de supprimer le premier attribut "N=1" :

```
C_BLOB(maVarBlob)
C_TEXTE($ref_XML_Parent;$ref_XML_Enfant)
C_ENTIER LONG($NumLigne)

$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire premier element XML enfant($ref_XML_Parent)
DOM SUPPRIMER ATTRIBUT XML($ref_XML_Enfant;"N")
```

DOM SUPPRIMER ELEMENT XML

DOM SUPPRIMER ELEMENT XML (refElément)

Paramètre	Type		Description
refElément	Chaîne	⇨	Référence d'élément XML

Description

La commande **DOM SUPPRIMER ELEMENT XML** supprime l'élément désigné par *refElément*.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.
Une erreur est générée lorsque la référence de l'élément n'est pas valide.

XML SAX

- Présentation des commandes XML SAX
- SAX AJOUTER CDATA XML
- SAX AJOUTER COMMENTAIRE XML
- SAX AJOUTER DOCTYPE XML
- SAX AJOUTER INSTRUCTION DE TRAITEMENT
- SAX AJOUTER VALEUR ELEMENT XML
- SAX FERMER ELEMENT XML
- SAX FIXER DECLARATION XML
- SAX LIRE CDATA XML
- SAX LIRE COMMENTAIRE XML
- SAX LIRE ELEMENT XML
- SAX LIRE ENTITE XML
- SAX LIRE INSTRUCTION DE TRAITEMENT XML
- SAX Lire noeud XML
- SAX LIRE VALEUR ELEMENT XML
- SAX LIRE VALEURS DOCUMENT XML
- SAX OUVRIR ELEMENT XML
- SAX OUVRIR ELEMENT XML TABLEAUX

Présentation des commandes XML SAX

Ce thème regroupe les commandes XML SAX de 4D.

Pour des informations générales sur le XML (présentation, jeux de caractères, glossaire) ainsi que sur les différences entre les modes DOM et SAX, veuillez vous reporter à la section **Présentation des commandes XML DOM**.

Note à propos du mode préemptif : Les références XML créées par un process préemptif peuvent être utilisées dans ce process uniquement. A l'inverse, les références XML créées par des process coopératifs peuvent être utilisées par tout autre process coopératif, mais ne peuvent pas être utilisées par un process préemptif.

Création, ouverture et fermeture des documents XML via SAX

Les commandes SAX travaillent avec des références de documents standard de 4D (*RefDoc*, référence de type Heure). Il est donc possible d'utiliser ces commandes conjointement avec les commandes 4D permettant de gérer les documents, par exemple **ENVOYER PAQUET** ou **Ajouter a document**.
La création et l'ouverture par programmation de documents XML est effectuée via les commandes **Creer document** et **Ouvrir document**. Par la suite, l'utilisation d'une commande XML avec ces documents provoquera la mise en oeuvre automatique des mécanismes XML tels que l'encodage. Par exemple, l'en-tête *<?xml version="1.0" encoding="… encodage …" standalone = "no "?>* sera automatiquement écrit dans le document.

Note : Les documents destinés à la lecture SAX doivent impérativement être ouverts en mode "lecture seule" par la commande **Ouvrir document**. Ce principe est destiné à prévenir les conflits pouvant survenir entre 4D et la librairie Xerces lors de l'ouverture simultanée de documents "standard" et de documents XML. Si vous exécutez une commande d'analyse SAX avec un document ouvert en lecture-écriture, un message d'alerte est affiché et l'analyse est impossible.

La fermeture d'un document XML SAX doit être effectuée à l'aide de la commande **FERMER DOCUMENT**. Si des éléments XML étaient ouverts, ils sont automatiquement refermés.

SAX AJOUTER CDATA XML

SAX AJOUTER CDATA XML (document ; données)

Paramètre	Type		Description
document	RefDoc	➡	Référence du document ouvert
données	BLOB, Texte	➡	Texte ou BLOB à insérer dans le document entre balises CData

Description

La commande **SAX AJOUTER CDATA XML** ajoute dans le document XML référencé par *document* des *données* de type texte ou BLOB. Ces données seront automatiquement encadrées par les balises *<![CDATA[* et *]]>*.
Le texte compris dans une section CData est ignoré par l'interpréteur XML.

Si vous souhaitez encoder le contenu de *données*, vous devez utiliser la commande **ENCODER BASE64**. Dans ce cas bien entendu, vous devez passer un BLOB dans *données*.

Pour que cette commande fonctionne, un élément doit être ouvert. Dans le cas contraire, une erreur est générée.

Exemple

Vous souhaitez insérer les lignes suivantes dans votre document XML :

```
function matchwo(a,b) { if (a < b && a < 0) then       {       return 1       } else       {       return 0
      } }
```

Pour cela, il vous suffit d'exécuter le code suivant :

```
C_TEXTE(vtMontexte)
... ` placez ici le texte dans la variable vtMontexte
SAX AJOUTER CDATA XML($RefDoc;vtMontexte)
```

Le résultat sera alors :

```
<![CDATA[ function matchwo(a,b) { if (a < b && a < 0) then    {    return 1    } else    {    return 0    } } ]]>
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

SAX AJOUTER COMMENTAIRE XML

SAX AJOUTER COMMENTAIRE XML (document ; commentaire)

Paramètre	Type		Description
document	RefDoc	⇨	Référence du document ouvert
commentaire	Chaîne	⇨	Commentaire à ajouter

Description

La commande **SAX AJOUTER COMMENTAIRE XML** ajoute un *commentaire* dans le document XML référencé par *document*.

Un commentaire XML est un texte dont le contenu ne sera pas analysé par l'interpréteur XML. Les commentaires XML sont encadrés par les caractères <!-- et -->.

Exemple

L'instruction suivante :

```
vCommentaire:="Créé par 4D"
SAX AJOUTER COMMENTAIRE XML($RefDoc;vCommentaire)
```

... inscrira cette ligne dans le document :

```
<!--Créé par 4D-->
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Gestion des erreurs

En cas d'erreur, la commande retourne une erreur qui peut être interceptée via une méthode d'appel sur erreur.

SAX AJOUTER DOCTYPE XML

SAX AJOUTER DOCTYPE XML (document ; docType)

Paramètre	Type		Description
document	RefDoc	⇨	Référence du document ouvert
docType	Chaîne	⇨	DocType à ajouter

Description

La commande **SAX AJOUTER DOCTYPE XML** ajoute l'instruction DocType définie par le paramètre *docType* dans le document XML référencé par *document*.

Une instruction DocType permet d'indiquer le type de XML dans lequel le document a été écrit et de désigner la Déclaration de type de document (DTD) utilisée. Une instruction DocType est généralement de la forme *<!DOCTYPE type_XML "adresse_DTD">*.

Exemple

L'instruction suivante :

```
vDocType:="Livres SYSTEM \"Livre.DTD\""
SAX AJOUTER DOCTYPE XML($RefDoc;vDocType)
```

... inscrira cette ligne dans le document :

```
<!DOCTYPE Livres SYSTEM"Livre.DTD">
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Gestion des erreurs

En cas d'erreur, la commande retourne une erreur qui peut être interceptée via une méthode d'appel sur erreur.

SAX AJOUTER INSTRUCTION DE TRAITEMENT

SAX AJOUTER INSTRUCTION DE TRAITEMENT (document ; instruction)

Paramètre	Type		Description
document	RefDoc	⇨	Référence du document ouvert
instruction	Texte	⇨	Instruction à insérer dans le document

Description

La commande **SAX AJOUTER INSTRUCTION DE TRAITEMENT** ajoute dans le document XML référencé par *document* une *instruction* de traitement XML.

Une instruction de traitement permet d'indiquer le type d'application et éventuellement des paramètres additionnels permettant de traiter une entité externe non analysable.

La commande formate les données d'instruction conformément au XML. En revanche, les instructions elles-mêmes ne sont pas analysées, il revient au développeur de s'assurer qu'elles sont valides.

Exemple

Le code suivant :

... inscrira cette ligne dans le document :

```
vtInstruct:="xml-stylesheet type="+Caractere(Guillemets)+"text/xsl"+Caractere(Guillemets)+"href="+
Caractere(Guillemets)+"style.xsl"+Caractere(Guillemets)
SAX AJOUTER INSTRUCTION DE TRAITEMENT($RefDoc;vtInstruct)
```

... inscrira cette ligne dans le document :

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX AJOUTER VALEUR ELEMENT XML

SAX AJOUTER VALEUR ELEMENT XML (document ; données {; *})

Paramètre	Type		Description
document	RefDoc	➡	Référence du document ouvert
données	Texte, Variable	➡	Texte ou variable à insérer dans le document
*	Opérateur	➡	Si passé = Encoder les caractères spéciaux en mode 'XML Données brutes'

Description

La commande **SAX AJOUTER VALEUR ELEMENT XML** ajoute directement dans le document XML référencé par *document* des *données* sans les convertir. Cette commande équivaut par exemple à insérer une pièce jointe dans le corps (body) d'un email.

Vous pouvez passer dans le paramètre *données* soit directement une chaîne de caractères, soit une variable 4D. Le contenu de la variable sera converti en texte pour pouvoir être inséré dans le document XML.
Si vous souhaitez encoder le contenu de *données*, vous devez utiliser la commande **ENCODER BASE64**. Dans ce cas bien entendu, vous devez passer un BLOB dans *données*.

Par défaut, la commande encode les caractères spéciaux (< > " '...) contenus dans le paramètre *données*, sauf si vous avez désactivé ce mécanisme pour le process courant à l'aide de la commande **XML FIXER OPTIONS** en passant la valeur XML données brutes à l'option XML encodage chaînes. Par exemple :

```
XML FIXER OPTIONS($refDoc;XML encodage chaînes;XML données brutes)
```

Dans ce contexte, pour forcer l'encodage des caractères spéciaux lors de l'appel de **SAX AJOUTER VALEUR ELEMENT XML**, il est nécessaire de passer le paramètre facultatif *.

Pour que cette commande fonctionne, un élément doit être ouvert. Dans le cas contraire, une erreur est générée.

Exemple

Cet exemple insère le fichier *whitepaper.pdf* dans l'élément XML ouvert :

```
C_BLOB(vBMonBLOB)
DOCUMENT VERS BLOB("c:\\whitepaper.pdf";vBMonBLOB)
SAX AJOUTER VALEUR ELEMENT XML($RefDoc;vBMonBLOB)
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX FERMER ELEMENT XML

SAX FERMER ELEMENT XML (document)

Paramètre	Type		Description
document	RefDoc	⇨	Référence du document ouvert

Description

La commande **SAX FERMER ELEMENT XML** inscrit dans le document XML référencé par *document* les instructions nécessaires à la fermeture du dernier élément ouvert via la commande **SAX OUVRIR ELEMENT XML**.

L'emploi de cette commande est facultatif. En effet, 4D ajoute automatiquement si nécessaire, au moment de la fermeture des documents XML, les balises de fin d'éléments non refermés explicitement.

Exemple

Si le dernier élément ouvert est <*Book*>, l'instruction suivante :

```
SAX FERMER ELEMENT XML($RefDoc)
```

... inscrira cette ligne dans le document :

```
</Book>
```

SAX FIXER DECLARATION XML

SAX FIXER DECLARATION XML (document ; encodage {; autonome {; indentation}})

Paramètre	Type		Description
document	RefDoc	⇒	Référence du document ouvert
encodage	Chaîne	⇒	Jeu de caractères du document XML
autonome	Booléen	⇒	Vrai=le document est autonome, Faux (défaut)=le document n'est pas autonome
indentation	Booléen	⇒	*** Obsolète, ne plus utiliser ***

Description

La commande **SAX FIXER DECLARATION XML** initialise le document XML référencé par *document* à l'aide des valeurs passées en paramètres. Ces paramètres permettent de déterminer l'encodage, l'attribut autonome (standalone) et l'indentation du document.

- *encodage* : indique le jeu de caractères employé dans le document. Par défaut (si la commande n'est pas appelée), le jeu de caractères UTF-8 (Unicode compressé) est utilisé.
  Note : Si vous passez un jeu de caractères non pris en charge par les commandes XML de 4D, l'UTF-8 sera utilisé. Reportez-vous au paragraphe **Jeux de caractères** pour connaître la liste des jeux de caractères pris en charge (l'UTF-8 est toutefois recommandé dans la plupart des cas).
- *autonome* : indique si le document est autonome (**Vrai**) ou s'il dépend, pour son fonctionnement, d'autres fichiers ou de ressources externes (**Faux**). Par défaut (si la commande n'est pas appelée ou si le paramètre est omis), le document n'est pas autonome.

Note de compatibilité : Le paramètre *indentation* est conservé pour des raisons de compatibilité avec les versions précédentes de 4D mais son usage est déconseillé à compter de 4D v12. Désormais, pour définir l'indentation du document, il est fortement recommandé d'utiliser la commande **XML FIXER OPTIONS**.

Cette commande doit être appelée une seule fois par document et avant la première commande d'écriture XML dans le document, sinon une erreur est générée.

Exemple

Le code suivant :

```
SAX FIXER DECLARATION XML($RefDoc;"UTF-16";Vrai)
```

... inscrira cette ligne dans le document :

```
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
```

SAX LIRE CDATA XML

SAX LIRE CDATA XML (document ; valeur)

Paramètre	Type		Description
document	RefDoc	⇒	Référence du document ouvert
valeur	Texte, BLOB	⇐	Valeur de l'élément

Description

La commande **SAX LIRE CDATA XML** permet de récupérer la *valeur* CDATA d'un élément XML existant dans le document XML référencé par *document*. Elle doit être appelée dans le contexte d'un événement SAX CDATA XML. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande **SAX Lire noeud XML**.

Passez une variable *valeur* de type Texte si vous souhaitez récupérer des données de taille supérieure à 32 Ko (la base doit fonctionner en mode Unicode).

Note de compatibilité : A compter de 4D v12, les contenus CDATA encodés en base64 sont automatiquement décodés par la commande **SAX LIRE CDATA XML**, il est donc inutile d'appeler la commande **DECODER BASE64**.

Exemple

Considérons l'extrait de code XML suivant :

```
<RootElement>    <Child>MonTexte<![CDATA[MonCData]]</Child> </RootElement>
```

Le code 4D suivant retournera "MonCData" dans *vDonnéesTexte* :

```
C_BLOB(vDonnées)
C_TEXTE(vDonnéesTexte)
SAX LIRE CDATA XML(RefDoc;vDonnées)
vDonnéesTexte:=BLOB vers texte(vDonnées;UTF8 chaîne en C)
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE COMMENTAIRE XML

SAX LIRE COMMENTAIRE XML (document ; commentaire)

Paramètre	Type		Description
document	RefDoc	⇨	Référence du document ouvert
commentaire	Chaîne	⇦	Commentaire XML

Description

La commande **SAX LIRE COMMENTAIRE XML** retourne un *commentaire* si un événement SAX de type XML commentaire est généré dans le document XML référencé par *document*. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande **SAX Lire noeud XML**.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE ELEMENT XML

SAX LIRE ELEMENT XML (document ; nom ; préfixe ; nomsAttributs ; valeursAttributs)

Paramètre	Type		Description
document	RefDoc	➡	Référence du document ouvert
nom	Chaîne	⬅	Nom de l'élément
préfixe	Chaîne	⬅	Espace de nommage
nomsAttributs	Tableau chaîne	⬅	Noms des attributs
valeursAttributs	Tableau chaîne	⬅	Valeurs des attributs

Description

La commande **SAX LIRE ELEMENT XML** retourne diverses informations relatives à l'élément *nom* présent dans le document XML référencé par *document*. Elle doit être appelée dans le contexte d'un événement SAX XML début élément ou XML fin élément. Dans le cas particulier d'un XML fin élément, les paramètres d'attributs ne sont pas gérés. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande **SAX Lire noeud XML**.

nom contient le nom de l'élément.

préfixe retourne l'espace de nommage (namespace) de l'élément. Ce paramètre est vide si aucun espace de nommage n'est associé à l'élément.

La commande remplit le tableau *nomsAttributs* avec les noms des attributs de l'élément cible. Si nécessaire, la commande crée et dimensionne automatiquement le tableau.

La commande remplit également le tableau *valeursAttributs* avec les valeurs des attributs de l'élément cible. Si nécessaire, la commande crée et dimensionne automatiquement le tableau.

Exemple

Considérons l'extrait de code XML suivant :

```
<RootElement>
<Child Att1="111"Att2="222"Att3="333">MonTexte</Child>
</RootElement>
```

Une fois l'instruction suivante exécutée :

```
SAX LIRE ELEMENT XML(RefDoc;vNom;vPréfixe;tAttrNoms;tAttrValeurs)
```

...*vNom* contiendra "Child"
vPréfixe contiendra ""
tAttrNoms{1} contiendra "Att1", *tAttrNoms{2}* contiendra "Att2", *tAttrNoms{3}* contiendra "Att3"
tAttrValeurs{1} contiendra "111", *tAttrValeurs{2}* contiendra "222", *tAttrValeurs{3}* contiendra "333"

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE ENTITE XML

SAX LIRE ENTITE XML (document ; nom ; valeur)

Paramètre	Type		Description
document	RefDoc	➡	Référence du document ouvert
nom	Chaîne	⬅	Nom de l'entité
valeur	Chaîne	⬅	Valeur de l'entité

Description

La commande **SAX LIRE ENTITE XML** permet de récupérer le *nom* et la *valeur* d'une entité XML présente dans le document XML référencé par *document*. Elle doit être appelée dans le contexte d'un événement SAX XML entité. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande **SAX Lire noeud XML**.

Exemple

Considérons l'extrait de code XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE body [ < <!ELEMENT body (element*)> <!ELEMENT element (#PCDATA)>
<!ENTITY nom "Le remplacement"> ]> <body> <element>L'entité est mise à jour par &nom; </body>
```

L'instruction suivante retournera "nom" dans *vNom* et "Le remplacement" dans *vValeur*.

```
SAX LIRE ENTITE XML(RefDoc;vNom;vValeur)
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE INSTRUCTION DE TRAITEMENT XML

SAX LIRE INSTRUCTION DE TRAITEMENT XML (document ; nom ; valeur)

Paramètre	Type		Description
document	RefDoc	⇨	Référence du document ouvert
nom	Chaîne	⇦	Nom de l'instruction
valeur	Chaîne	⇦	Valeur de l'instruction

Description

La commande **SAX LIRE INSTRUCTION DE TRAITEMENT XML** retourne le *nom* et la *valeur* de l'instruction de traitement XML analysée dans le document XML référencé par *document*. Cette commande doit être appelée dans le contexte d'un événement XML instruction de traitement. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande **SAX Lire noeud XML**.

Exemple

Considérons l'extrait de code XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?> <!-- Edited with XML Spy v3.0.7 NT (http://www.xmlspy.com) by Myself (4D SA)--> <?PI TextProcess?> <!DOCTYPE RootElement SYSTEM "ParseTest.dtd">
```

L'instruction suivante retournera "PI" dans *vNom* et "TextProcess" dans *vValeur* :

```
SAX LIRE INSTRUCTION DE TRAITEMENT XML($RefDoc;vNom;vValeur)
```

SAX Lire noeud XML

SAX Lire noeud XML (document) -> Résultat

Paramètre	Type		Description
document	RefDoc	⇨	Référence du document ouvert
Résultat	Entier long	⇦	Evénement retourné par la fonction

Description

La commande **SAX Lire noeud XML** retourne un entier long indiquant le type d'événement SAX retourné durant l'analyse du document XML référencé par *document*.

Les événements pouvant être retournés sont fournis sous forme de constantes dans le thème "**XML**" :

Constante	Type	Valeur
XML CDATA	Entier long	7
XML commentaire	Entier long	2
XML début document	Entier long	1
XML début élément	Entier long	4
XML Donnée	Entier long	6
XML entité	Entier long	8
XML fin document	Entier long	9
XML fin élément	Entier long	5
XML instruction de traitement	Entier long	3

Exemple

Exemple de traitement des événements :

```
RefDoc:=Ouvrir document("";"xml";Mode lecture) `Ouverture en lecture seule obligatoire
Si(OK=1)
   Repeter
      MonEvénement:=SAX Lire noeud XML(RefDoc)
      Au cas ou
         :(MonEvénement=XML début document)
            FaireQuelqueChose
         :(MonEvénement=XML commentaire)
            FaireAutreChose
      Fin de cas
   Jusque(MonEvénement=XML fin document)
   FERMER DOCUMENT(RefDoc)
Fin de si
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE VALEUR ELEMENT XML

SAX LIRE VALEUR ELEMENT XML (document ; valeur)

Paramètre	Type		Description
document	RefDoc	➡	Référence du document ouvert
valeur	Texte, BLOB	⬅	Valeur de l'élément

Description

La commande **SAX LIRE VALEUR ELEMENT XML** permet de récupérer la *valeur* d'un élément XML existant dans le document XML référencé par *document*. Elle doit être appelée dans le contexte d'un événement SAX XML Donnée. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande **SAX Lire noeud XML**.

Passez dans le paramètre *valeur* une variable de type Texte ou BLOB devant récupérer les données. Si vous passez un BLOB, la commande tentera automatiquement de le décoder en base64.

Exemple

Considérons l'extrait de code XML suivant :

```
<RootElement>    <Child Att1="111" Att2="222" Att3="333">MonTexte</Child> </RootElement>
```

L'instruction suivante retournera "MonTexte" dans *vValeur* :

```
SAX LIRE VALEUR ELEMENT XML(RefDoc;vValeur)
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE VALEURS DOCUMENT XML

SAX LIRE VALEURS DOCUMENT XML (document ; encodage ; version ; autonome)

Paramètre	Type		Description
document	RefDoc	⇨	Référence du document ouvert
encodage	Chaîne	⇦	Jeu de caractères du document XML
version	Chaîne	⇦	Version du XML
autonome	Booléen	⇦	Vrai=le document est autonome, sinon Faux

Description

La commande **SAX LIRE VALEURS DOCUMENT XML** extrait des informations élémentaires de l'en-tête XML du document XML référencé par *document*.

La commande retourne respectivement le type d'encodage, la version et la propriété "autonome" du document dans les paramètres *encodage*, *version* et *autonome*. Cette commande doit être utilisée dans le contexte de l'événement SAX XML début document. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande **SAX Lire noeud XML**.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX OUVRIR ELEMENT XML

SAX OUVRIR ELEMENT XML (document ; balise {; nomAttribut ; valeurAttribut} {; nomAttribut2 ; valeurAttribut2 ; ... ; nomAttributN ; valeurAttributN})

Paramètre	Type		Description
document	RefDoc	➡	Référence du document ouvert
balise	Chaîne	➡	Nom de l'élément à ouvrir
nomAttribut	Chaîne	➡	Nom d'attribut
valeurAttribut	Chaîne	➡	Valeur d'attribut

Description

La commande **SAX OUVRIR ELEMENT XML** permet d'ajouter un nouvel élément dans le document XML référencé par *document* ainsi que, facultativement, des attributs et leurs valeurs.

L'élément ajouté est "ouvert" dans le document (la balise de fin n'est pas ajoutée). Pour refermer un élément créé à l'aide de cette commande, vous devez soit :

- utiliser la commande **SAX FERMER ELEMENT XML**,
- refermer le document XML. Dans ce cas, 4D ajoute automatiquement les balises XML de fermeture nécessaires.

Passez dans *balise* le nom de l'élément à créer. Ce nom peut contenir uniquement des lettres, des chiffres, ainsi que les caractères ".", "-","_" et ":". Si un caractère invalide est passé dans *balise*, une erreur est générée.

Facultativement, la commande permet de passer un ou plusieurs couple(s) attributs/valeurs (sous forme de variables, champs ou valeur littérales) via les paramètres *nomAttribut* et *valeurAttribut*. Vous pouvez passer autant de couples attribut/valeur que vous voulez.

Exemple

L'instruction suivante :

```
vElement:="Book"
SAX OUVRIR ELEMENT XML($RefDoc;vElement)
```

... inscrira cette ligne dans le document :

```
<Book
```

Gestion des erreurs

Si un caractère invalide est passé dans *balise*, une erreur est générée.

SAX OUVRIR ELEMENT XML TABLEAUX

SAX OUVRIR ELEMENT XML TABLEAUX (document ; balise {; tabNomsAttributs ; tabValeursAttributs} {; tabNomsAttributs2 ; tabValeursAttributs2 ; ... ; tabNomsAttributsN ; tabValeursAttributsN})

Paramètre	Type		Description
document	RefDoc	➡	Référence du document ouvert
balise	Chaîne	➡	Nom de l'élément à ouvrir
tabNomsAttributs	Tableau chaîne	➡	Tableau de noms d'attributs
tabValeursAttributs	Tableau chaîne, Tableau entier long, Tableau date, Tableau réel, Tableau image, Tableau booléen	➡	Tableau de valeurs d'attributs

Description

La commande **SAX OUVRIR ELEMENT XML TABLEAUX** permet d'ajouter un nouvel élément dans le document XML référencé par *document* ainsi que, facultativement, des attributs et leurs valeurs sous forme de tableaux.

Hormis la prise en charge de tableaux (cf. ci-dessous), cette commande est identique à **SAX OUVRIR ELEMENT XML**. Reportez-vous à la description de cette commande pour le détail de son fonctionnement.

SAX OUVRIR ELEMENT XML TABLEAUX accepte des tableaux de type date, numérique, booléen et image comme paramètre(s) *tabValeursAttributs*. 4D effectue automatiquement les conversions nécessaires, vous pouvez paramétrer ces conversions à l'aide de la commande **XML FIXER OPTIONS**.

Facultativement, la commande **SAX OUVRIR ELEMENT XML TABLEAUX** permet de passer plusieurs couples d'attributs et de valeurs d'attributs sous forme de tableaux dans les paramètres *tabNomsAttributs* et *tabValeursAttributs*.
Les tableaux doivent avoir été créés au préalable et fonctionner par paires. Vous pouvez passer autant de couples de tableaux et autant d'éléments dans chaque couple que vous voulez.

Exemple

La méthode suivante :

```
TABLEAU ALPHA(80;tNomsAtt;2)
TABLEAU ALPHA(80;tValeursAtt;2)
vElement:="Book"
tNomsAtt{1}:="Font"
tValeursAtt{1}:="arial"
tNomsAtt{2}:="Style"
tValeursAtt{2}:="Bold"
SAX OUVRIR ELEMENT XML TABLEAUX($RefDoc;vElement;tNomsAtt;tValeursAtt)
```

... inscrira cette ligne dans le document :

```
<Book Font="arial" Style="Bold">
```

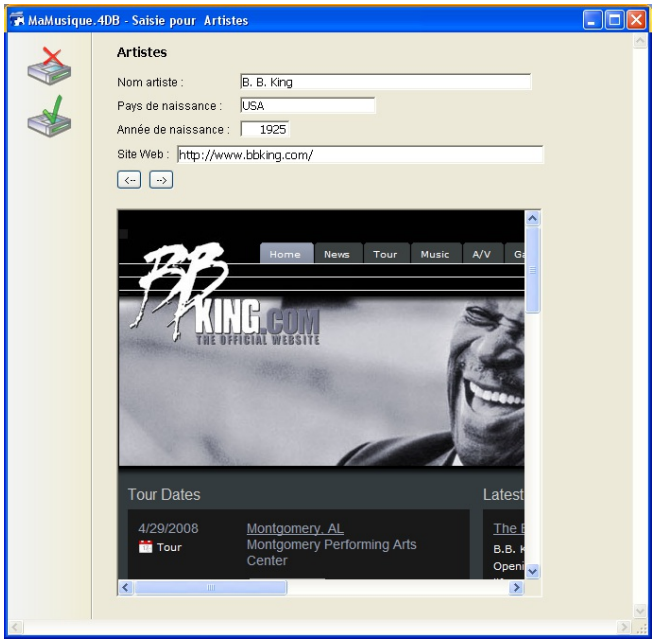
Zone Web

- Gestion programmée des zones Web
- WA ACTUALISER URL
- WA AGRANDIR TEXTE PAGE
- WA ARRETER CHARGEMENT URL
- WA Creer menu historique URL
- WA Evaluer JavaScript
- WA EXECUTER FONCTION JAVASCRIPT
- WA FIXER CONTENU PAGE
- WA FIXER FILTRES LIENS EXTERNES
- WA FIXER FILTRES URL
- WA FIXER PREFERENCE
- WA Lire contenu page
- WA Lire dernier URL filtre
- WA LIRE DERNIERE ERREUR URL
- WA LIRE FILTRES LIENS EXTERNES
- WA LIRE FILTRES URL
- WA LIRE HISTORIQUE URL
- WA LIRE PREFERENCE
- WA Lire titre page
- WA Lire URL courant
- WA OUVRIR URL
- WA OUVRIR URL PRECEDENT
- WA OUVRIR URL SUIVANT
- WA REDUIRE TEXTE PAGE
- WA URL precedent disponible
- WA URL suivant disponible

Gestion programmée des zones Web

Les commandes de ce thème sont dédiées à la gestion programmée des objets de formulaire de type Zone Web (Web Area).

Les zones Web peuvent afficher tout type de contenu Web(*) à l'intérieur même de votre environnement 4D : pages HTML au contenu statique ou dynamique, fichiers, images, Javascript... L'image suivante montre une zone Web incluse dans un formulaire et affichant une page HTML :



(*) L'usage de plugins Web et d'applets Java est toutefois déconseillé (cf. **Notes d'utilisation des zones Web**).

Outre les commandes du thème Zone Web, plusieurs actions standard et des événements formulaires dédiés permettent au développeur de contrôler le fonctionnement des zones Web. Des variables spécifiques prennent en charge l'interaction entre la zone Web et l'environnement 4D. Ces outils vous permettent ainsi de développer un navigateur Web basique dans vos formulaires.

Créer et adresser une zone Web

La création d'une zone Web s'effectue à l'aide d'une variante du bouton Zone de plug-in/Sous-formulaire de la barre d'objets de l'éditeur de formulaires de 4D (pour plus d'information, reportez-vous à la section **Zones Web** dans le manuel *Mode Développement*).

Note : Lorsqu'une zone Web utilisant le moteur de rendu intégré est affichée dans un process créé avec la commande **Nouveau process**, il est recommandé d'utiliser la valeur par défaut (0) pour le paramètre *pile*.

Comme les autres objets dynamiques de formulaire, une zone Web dispose d'un nom d'objet et d'un nom de variable, vous permettant de l'adresser par programmation. La variable standard associée à l'objet zone Web est de type Texte. Vous pouvez en particulier utiliser les commandes **OBJET FIXER VISIBLE** et **OBJET DEPLACER** avec les zones Web.

Note : La variable Texte associée à la zone Web ne contient pas de référence et ne peut donc pas être passée en tant que paramètre à une méthode. Par exemple, pour une zone Web nommée **MaZone**, le code suivant ne peut pas être utilisé :

```
Mamethode(MaZone)
```

Code de *Mamethode* :

```
WA ACTUALISER URL($1) //Ne fonctionne pas
```

Pour ce type de programmation, vous devez utiliser des pointeurs :

```
Mamethode(->MaZone)
```

Code de *Mamethode* :

```
WA ACTUALISER URL($1->) //Fonctionne
```

Gestion des variables associées

Outre la variable objet standard (cf. paragraphe précédent), deux variables spécifiques sont automatiquement associées à chaque zone Web :

- la variable "URL"
- la variable "Progression"

Vous pouvez nommer ces variables comme vous le souhaitez. Les variables sont accessibles dans la Liste des propriétés :

Variable URL

La variable "URL" est de type chaîne. Elle contient l'URL chargé ou en cours de chargement par la zone Web associée.
L'association entre la variable et la zone Web s'effectue dans les deux sens :

- Si l'utilisateur affecte un nouvel URL à la variable, l'URL est automatiquement chargé par la zone Web.
- Toute navigation effectuée à l'intérieur de la zone Web met automatiquement à jour le contenu de la variable.
  Schématiquement, cette variable fonctionne comme la zone d'adresse d'un navigateur Web. Vous pouvez la représenter par une zone de texte située au-dessus de la zone Web.

Variable URL et commande WA OUVRIR URL
La variable URL produit les mêmes effets que la commande **WA OUVRIR URL**. Les différences suivantes sont toutefois à noter :

- pour les accès aux documents, la variable accepte uniquement des URLs conformes aux RFC ("file://c:/Mon%20Doc") et non les chemins d'accès système ("c:\MonDoc"). La commande **WA OUVRIR URL** accepte les deux notations.
- si la variable URL contient une chaîne vide, la zone Web ne tente pas de charger l'URL. La commande **WA OUVRIR URL** génère une erreur dans ce cas.
- si la variable URL ne contient pas de protocole (http, mailto, file, etc.), la zone Web ajoute "http://", ce qui n'est pas le cas pour la commande **WA OUVRIR URL**.
- lorsque la zone Web n'est pas affichée dans le formulaire (lorsqu'elle se trouve sur une autre page du formulaire), l'exécution de la commande **WA OUVRIR URL** est sans effet tandis que la valorisation de la variable URL permet de mettre à jour l'URL courant.

Variable Progression (du chargement)

La variable "Progression" est de type Entier long. Elle contient une valeur entre 0 et 100, représentant le pourcentage du chargement complet de la page affichée dans la zone Web.
La variable est mise à jour automatiquement par 4D. Il n'est pas possible de la modifier manuellement.

Accéder aux méthodes 4D

Il est possible d'appeler des méthodes 4D depuis le code JavaScript exécuté dans une zone Web et de recevoir des valeurs en retour.

Important : Cette fonction est disponible uniquement si la zone Web utilise le moteur de rendu Web intégré.

Configurer la zone Web

Pour pouvoir appeler des méthodes 4D depuis la zone Web, vous devez cocher l'option **Accès méthodes 4D** pour la zone dans la Liste des propriétés :

Note : Cette option n'apparaît que si l'option **Utiliser le moteur de rendu Web intégré** est cochée.

Lorsque cette propriété est cochée, un objet JavaScript spécial ($4d) est instancié dans la zone Web et permet de gérer les appels aux méthodes projet de 4D.

Utilisation de l'objet $4d

Lorsque l'option **Accès méthodes 4D** est cochée, le moteur de rendu Web intégré de 4D fournit à la zone un objet JavaScript nommé $4d que vous pouvez associer à toute méthode projet 4D à l'aide de la notation objet ".".

Par exemple, pour appeler la méthode 4D ***HelloWorld***, il vous suffit d'exécuter l'instruction JavaScript :

```
$4d.HelloWorld();
```

Attention : Notez bien que l'objet est nommé $4d (le "d" est en minuscule). JavaScript tient compte de la casse des caractères.

La syntaxe des appels aux méthodes 4D est la suivante :

```
$4d.NomMethode4D(param1,paramN,function(result){})
```

- *param1...paramN* : Vous pouvez passer autant de paramètres que nécessaire à la méthode 4D.
  Vous pouvez passer tout type de paramètre pris en charge par JavaScript (chaîne, nombre, tableau, objet).

- *function(result)* : Fonction à passer en dernier argument. Cette fonction "callback" sera appelée en asynchrone une fois que la méthode 4D aura terminé son exécution. Elle recevra le paramètre *result*.
  - *result* : Résultat de l'exécution de la méthode 4D, retourné dans l'expression "$0".
    Ce résultat peut être de tout type pris en charge par JavaScript (chaîne, nombre, tableau, objet). Vous pouvez utiliser la commande **C_OBJET** pour retourner des objets.
    Note : Par défaut, 4D travaille en UTF-8. Lorsque vous retournez du texte contenant des caractères étendus, par exemple des caractères accentués, assurez-vous que l'encodage de la page affichée dans la zone Web est bien déclaré en UTF-8, sinon des caractères pourront être incorrectement rendus. Dans ce cas, ajoutez la ligne suivante dans la page HTML afin de déclarer l'encodage :
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

Exemple 1

Soit une méthode projet 4D nommée ***today*** ne recevant pas de paramètre et retournant la date courante sous forme de chaîne.

Code 4D de la méthode ***today*** :

```
C_TEXTE($0)
$0:=Chaine(Date du jour;Système date long)
```

Dans la zone Web, la méthode 4D peut être appelée avec la syntaxe suivante :

```
$4d.today()
```

La méthode 4D ne reçoit aucun paramètre, en revanche elle retourne la valeur de *$0* à la fonction de rétro-appel (*callback*) appelée par 4D à l'issue de l'exécution de la méthode. On souhaite afficher la date dans la page HTML qui sera chargée par la zone Web.

Voici le code de la page HTML :

```
<html> <head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /> <script
```

```
type="text/javascript"> $4d.today(function(dollarZero) {     var curDate = dollarZero;
document.getElementById("madiv").innerHTML=curDate; }); </script> </head> <body>Aujourd'hui nous sommes le : <div
id="madiv"></div> </body> </html>
```

Exemple 2

La méthode projet 4D ***calcSum*** reçoit des paramètres ($1...$n) et en retourne la somme dans $0 :

Code 4D de la méthode ***calcSum*** :

```
C_REEL(${1}) //reçoit n paramètres de type REEL
C_REEL($0) //retourne un réel
C_ENTIER LONG($i;$n)
$n:=Nombre de paramètres
Boucle($i;1;$n)
   $0:=$0+${$i}
Fin de boucle
```

Le code JavaScript exécuté dans la zone Web est donc :

```
$4d.calcSum(33, 45, 75, 102.5, 7, function(dollarZero)     {         var result = dollarZero // result vaut 262.5     });
```

Evénements formulaire

Des événements formulaire spécifiques sont destinés à la gestion programmée des zones Web, concernant notamment l'activation des liens :

- Sur début chargement URL
- Sur chargement ressource URL
- Sur fin chargement URL
- Sur erreur chargement URL
- Sur filtrage URL
- Sur ouverture lien externe
- Sur refus ouverture fenêtre

En outre, les zones Web prennent en charge les événements formulaire génériques suivants :

- Sur chargement
- Sur libération
- Sur gain focus
- Sur perte focus

Pour plus d'informations sur ces événements, reportez-vous à la description de la commande **Evenement formulaire**.

Accès à l'inspecteur Web

Il est possible d'afficher et d'utiliser un inspecteur Web au sein des zones Web de vos formulaires. L'inspecteur Web est un débogueur, proposé par les moteur de rendu Web intégré, permettant d'analyser le code et les flux d'information des pages Web.

Afficher l'inspecteur Web

Pour que vous puissiez afficher l'inspecteur Web dans une zone Web, les conditions suivantes doivent être réunies :

- le moteur de rendu Web intégré doit être sélectionné pour la zone (l'inspecteur Web n'est disponible que dans cette configuration) (cf. **Utiliser le moteur de rendu Web intégré**)
- le menu contextuel de la zone doit être activé (l'appel de l'inspecteur est effectué via ce menu) (cf. **Menu contextuel**)
- l'usage de l'inspecteur doit être expressément autorisé pour la zone à l'aide de l'instruction suivante :

  ```
  WA FIXER PREFERENCE(*;"WA";WA autoriser inspecteur Web;Vrai)
  ```

  Pour plus d'informations, reportez-vous à la description de la commande **WA FIXER PREFERENCE**.

Utilisation de l'Inspecteur Web

Lorsque les paramétrages décrits ci-dessus sont effectués, vous disposez de nouvelles options telles que **Inspect Element** dans le menu contextuel de la zone :

Lorsque vous sélectionnez cette option, le débogueur de la zone Web est alors affiché.

L'inspecteur Web est inclus dans le moteur de rendu intégré. Pour une description détaillée des fonctionnalités de ce débogueur, veuillez vous reporter à la documentation fournie du moteur de rendu Web utilisé.

Notes d'utilisation des zones Web

Interface utilisateur

Lors de l'exécution du formulaire, l'utilisateur dispose des fonctions d'interface standard des navigateurs dans la zone Web, ce qui lui permet d'interagir avec les autres zones du formulaire :

- commandes du menu **Edition** : lorsque la zone Web a le focus, les commandes du menu **Edition** permettent d'effectuer les actions de copier, coller, tout sélectionner, etc., en fonction de la sélection.
- menu contextuel : il est possible d'associer un menu contextuel standard à la zone Web via la Liste des propriétés (cf. **Menu contextuel**). L'affichage de ce menu peut également être contrôlé via la commande **WA FIXER PREFERENCE**).

- glisser-déposer : l'utilisateur peut effectuer des glisser-déposer de textes, d'images ou de documents à l'intérieur d'une zone Web ou entre une zone Web et les objets des formulaires 4D, en fonction des propriétés des objets 4D.
  Pour des raisons de sécurité, le changement du contenu d'une zone Web via le glisser-déposer d'un fichier ou d'un URL n'est pas autorisé par défaut à compter de 4D v14 R2. Dans ce cas, le curseur de la souris affiche une icône d'interdiction  . La possibilité de déposer des URLs ou des fichiers dans la zone doit être explicitement autorisée à l'aide de la commande **WA FIXER PREFERENCE**.

Conflit Zone Web et serveur Web (Windows)

Sous Windows, il est déconseillé d'accéder via une zone Web au serveur Web de l'application 4D contenant la zone car cette configuration peut provoquer un conflit paralysant l'application. Bien entendu, un 4D distant peut accéder au serveur Web du 4D Server, mais pas à son propre serveur Web.

Plugins Web et applets Java

L'usage de plugins Web et d'applets Java dans les zones Web **est déconseillé** car ils peuvent entraîner des instabilités dans le fonctionnement de 4D, notamment au niveau de la gestion des événements.

Insertion du protocole (Mac OS)

Les URLs manipulés par programmation dans les zones Web sous Mac OS doivent débuter par le protocole. Par exemple, vous devez passer la chaîne "http://www.monsite.fr" et non uniquement "www.monsite.fr".

WA ACTUALISER URL

WA ACTUALISER URL ({* ;} objet)

Paramètre	Type		Description
*	Opérateur	⇨	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇨	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande **WA ACTUALISER URL** provoque le rechargement de l'URL courant affiché dans la zone Web désignée par les paramètres * et *objet*.

WA AGRANDIR TEXTE PAGE

WA AGRANDIR TEXTE PAGE ({* ;} objet)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande **WA AGRANDIR TEXTE PAGE** augmente la taille du texte affiché dans la zone Web désignée par les paramètres * et *objet*.
Sous Mac OS, la portée de cette commande est la session 4D : le paramétrage effectué n'est pas conservé après la fermeture de l'application 4D.
Sous Windows, la portée de cette commande est globale : le paramétrage est conservé après la fermeture de l'application 4D.

WA ARRETER CHARGEMENT URL

WA ARRETER CHARGEMENT URL ({* ;} objet)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande **WA ARRETER CHARGEMENT URL** stoppe le chargement des ressources de l'URL courant de la zone Web désignée par les paramètres * et *objet*.

WA Creer menu historique URL

WA Creer menu historique URL ({* ;} objet {; direction}) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
direction	Entier long	⇒	0 ou omis=Liste des URLs précédents, 1=Liste des URLs suivants
Résultat	RefMenu	⇐	Référence du menu

Description

La commande **WA Creer menu historique URL** crée et remplit un menu pouvant être utilisé directement pour la navigation parmi les URLs visités au cours de la session dans la zone Web désignée par les paramètres * et *objet*. Elle permet de construire une interface de navigation personnalisée.

Les informations fournies concernent la session, c'est-à-dire la navigation effectuée dans une même zone Web tant que le formulaire n'a pas été refermé.

Passez dans *direction* une valeur indiquant la liste à récupérer. Vous pouvez utiliser l'une des constantes suivantes, placées dans le thème "**Zone Web**" :

Constante	Type	Valeur
WA URLs précédents	Entier long	0
WA URLs suivants	Entier long	1

Si vous omettez le paramètre *direction*, la valeur 0 est utilisée.

Une fois le menu généré, vous pouvez l'afficher via la commande de 4D **Pop up menu dynamique** et le manipuler via les commandes standard de gestion des menus de 4D. La chaîne retournée par la commande **Pop up menu dynamique** contient l'URL de la page visitée (voir exemple).

Appelez la commande **EFFACER MENU** pour supprimer un menu historique d'URL lorsqu'il est devenu inutile.

Exemple

Le code suivant pourrait être associé à un bouton 3D avec pop up menu libellé "Précédent" :

```
Au cas ou
   :(Evenement formulaire=Sur clic) //Clic simple
      WA OUVRIR URL PRECEDENT(WA_zone)
   :(Evenement formulaire=Sur clic alternatif) //Clic sur la flèche -> affichage du pop up
 //Créer un menu historique précédent
      $Menu:=WA Creer menu historique URL(WA_zone;WA URLs précédents)
 //Afficher ce menu dans un pop up
      $URL:=Pop up menu dynamique($Menu)
      Si($URL#"") //Si une ligne est sélectionnée
         WA OUVRIR URL(WA_zone;$URL) // Ouvrir la page Web
      Fin de si
      EFFACER MENU($Menu) //Effacer le menu pour libérer la mémoire
Fin de cas
```

WA Evaluer JavaScript

WA Evaluer JavaScript ({* ;} objet ; codeJS {; type}) -> Résultat

Paramètre	Type		Description
*	Opérateur	➡	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	➡	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
codeJS	Chaîne	➡	Code JavaScript
type	Entier long	➡	Type dans lequel convertir le résultat
Résultat	Date, Heure, Objet, Pointeur, Réel, Texte	⮌	Résultat de l'exécution

Description

La commande **WA Evaluer JavaScript** exécute dans la zone Web désignée par les paramètres * et *objet* le code JavaScript passé dans *codeJS* et retourne le résultat.

Par défaut, la commande retourne le résultat sous forme de chaîne. Vous pouvez toutefois préciser le typage de la valeur retournée à l'aide du paramètre optionnel *type*. Pour cela, vous pouvez passer dans *type* une des constantes suivantes, placées dans le thème "**Types champs et variables**" :

Constante	Type	Valeur
Est un booléen	Entier long	6
Est un entier long	Entier long	9
Est un numérique	Entier long	1
Est un objet	Entier long	38
Est un texte	Entier long	2
Est une date	Entier long	4
Est une heure	Entier long	11

Exemple 1

Cet exemple de code JavaScript provoque l'affichage de l'url précédent :

```
$résultat:=WA Evaluer JavaScript(MaZoneW;"history.back()")
```

Exemple 2

Cet exemple montre quelques évaluations avec conversion des valeurs reçues.

Des fonctions JavaScript sont placées dans un fichier html :

```
<!DOCTYPE html> <html>     <head>         <script>         function evalLong(){             return 123;         }
        function evalTexte(){             return "456";         }         function evalObjet(){
return {a:1,b:"hello world"};         }         function evalDate(){             return new Date();         }
</script>     </head>     <body>         TEST PAGE     </body> </html>
```

Vous écrivez dans la méthode du formulaire 4D :

```
Si(Evenement formulaire=Sur chargement)
   WA OUVRIR URL(*;"Web Area";"C:\\myBase\\index.html")
Fin de si
```

Vous pouvez alors évaluer le code JavaScript depuis 4D :

```
$Eval1:=WA Evaluer JavaScript(*;"Web Area";"evalLong()";Est un entier long)
    //$Eval1 = 123
    //$Eval1 = "123" si le type est omis
$Eval2:=WA Evaluer JavaScript(*;"Web Area";"evalTexte()";Est un texte)
    //$Eval2 = "456"
$Eval3:=WA Evaluer JavaScript(*;"Web Area";"evalObjet()";Est un objet)
    //$Eval3 = {"a":1,"b":"hello world"}
$Eval4:=WA Evaluer JavaScript(*;"Web Area";"evalDate()";Est une date)
    //$Eval4 = 21/06/13
    //$Eval4 = "2013-06-21T14:45:09.694Z" si le type est omis
```

WA EXECUTER FONCTION JAVASCRIPT

WA EXECUTER FONCTION JAVASCRIPT ({* ;} objet ; fonctionJS ; résultat | * {; param}{; param2 ; ... ; paramN})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
fonctionJS	Chaîne	⇒	Nom de la fonction JavaScript à exécuter
résultat | *	Variable	⇒	* pour une fonction sans résultat ou
		⇐	Résultat de la fonction (si attendu)
param	Chaîne, Numérique, Date, Objet	⇒	Paramètre(s) à passer à la fonction

Description

La commande **WA EXECUTER FONCTION JAVASCRIPT** exécute dans la zone Web désignée par les paramètres * et *objet* la fonction JavaScript *fonctionJS* et retourne facultativement son résultat dans le paramètre *résultat*.

Si la fonction ne retourne pas de résultat, passez * dans le paramètre *résultat*.

Vous pouvez passer dans *param* un ou plusieurs paramètre(s) contenant les paramètres de la fonction.

La commande prend en charge plusieurs types de paramètres aussi bien en entrée (*param*) qu'en sortie (*résultat*). Vous pouvez passer et récupérer des données de type numérique, date, objet et chaîne.

Exemple 1

Appel d'une fonction JavaScript avec 3 paramètres :

```
$JavaScriptFunction:="TheFunctionToBeExecuted"
$Param1:="10"
$Param2:="true"
$Param3:="1,000.2" `notez "," comme séparateur de milliers et "." comme séparateur décimal

WA EXECUTER FONCTION JAVASCRIPT(MaZoneW;$JavaScriptFunction;$Result;$Param1;$Param2;$Param3)
```

Exemple 2

La fonction JavaScript "getCustomerInfos" reçoit un identifiant numérique en paramètre et retourne un objet :

```
C_OBJET($Result)
C_ENTIER LONG($ID)
$ID:=1000
WA EXECUTER FONCTION JAVASCRIPT(*;"WA";"getCustomerInfos";$Result;$ID)
```

WA FIXER CONTENU PAGE

WA FIXER CONTENU PAGE ({* ;} objet ; contenu ; baseURL)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
contenu	Chaîne	⇒	Code HTML source
baseURL	Chaîne	⇒	URL pour les références relatives (Mac OS)

Description

La commande **WA FIXER CONTENU PAGE** remplace la page affichée dans la zone Web désignée par les paramètres * et *objet* par le code HTML passé dans le paramètre *contenu*.

Le paramètre *baseURL* permet de définir sous Mac OS un URL de base qui sera ajouté devant les liens relatifs éventuellement présents dans la page. Sous Windows, ce paramètre est sans effet, l'URL de base est indéfini. Il n'est donc pas possible d'utiliser des références relatives sur cette plate-forme.

Note : Sous Windows, il est impératif qu'une page ait déjà été chargée dans la zone Web avant que cette commande puisse être appelée. Si nécessaire, vous pouvez passer l'URL "about:blank" afin de charger une page blanche.

Exemple

Affichage de la phrase "Hello world !" et définition d'un URL de base "file:///" (Mac OS uniquement) :

```
WA FIXER CONTENU PAGE(MaZoneW;"<html><body><h1>Hello World!</h1></body></html>";"file:///")
```

WA FIXER FILTRES LIENS EXTERNES

WA FIXER FILTRES LIENS EXTERNES ({* ;} objet ; tabFiltres ; tabAutorisRefus)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabFiltres	Tableau chaîne	⇒	Tableau de filtres
tabAutorisRefus	Tableau booléen	⇒	Tableau autoriser-refuser

Description

La commande **WA FIXER FILTRES LIENS EXTERNES** permet de mettre en place un ou plusieurs filtre(s) de liens externes pour la zone Web désignée par les paramètres * et *objet*. Les filtres de liens externes déterminent si un URL associé à la page courante via un lien doit être ouvert dans la zone Web ou dans le navigateur Web par défaut de la machine.

Lorsque l'utilisateur clique sur un lien dans la page courante, 4D consulte la liste des filtres de liens externes afin de vérifier si l'URL demandé doit être ouvert dans le navigateur de la machine. Si c'est le cas, la page correspondant à l'URL est affichée dans le navigateur Web et l'événement formulaire Sur ouverture lien externe est généré (cf. commande **Evenement formulaire**). Sinon (fonctionnement par défaut), la page correspondant à l'URL est affichée dans la zone Web. L'évaluation de l'URL est basée sur le contenu des tableaux *tabFiltres* et *tabAutorisRefus*.

Les tableaux *tabFiltres* et *tabAutorisRefus* doivent être synchronisés.

- Chaque ligne du tableau *tabFiltres* doit contenir un URL devant être filtré. Vous pouvez utiliser le * comme joker de remplacement pour un ou plusieurs caractère(s).
- Chaque ligne correspondante dans le tableau *tabAutorisRefus* doit contenur un booléen indiquant si l'URL doit être ouvert dans la zone Web (**Vrai**) ou dans le navigateur Web (**Faux**).

En cas de contradiction au niveau des paramétrages (autorisation et refus d'un même URL), le paramétrage pris en compte est le dernier.

Pour désactiver le filtrage des URLs, appelez la commande en lui passant des tableaux vides ou en passant respectivement les valeurs "*" et **Vrai** dans la dernière ligne des tableaux *tabFiltres* et *tabAutorisRefus*.

Important : Le filtrage établi par la commande **WA FIXER FILTRES URL** est pris en compte avant celui de **WA FIXER FILTRES LIENS EXTERNES**. Cela signifie que si un URL est refusé à cause d'un filtre de la commande **WA FIXER FILTRES URL**, il ne pourra pas être ouvert dans un navigateur même s'il est explicitement défini par la commande **WA FIXER FILTRES LIENS EXTERNES** (cf. exemple 2).

Exemple 1

Cet exemple provoquera l'ouverture de sites dans des navigateurs externes :

```
TABLEAU ALPHA(0;$filters;0)
TABLEAU BOOLEEN($AllowDeny;0)

AJOUTER A TABLEAU($filters;"*www.google.*") `Sélectionner "google"
AJOUTER A TABLEAU($AllowDeny;Faux) `Faux : ce lien sera ouvert dans un navigateur externe
AJOUTER A TABLEAU($filters;"*www.apple.*")
AJOUTER A TABLEAU($AllowDeny;Faux) `Faux : ce lien sera ouvert dans un navigateur externe
WA FIXER FILTRES LIENS EXTERNES(MaZoneW;$filters;$AllowDeny)
```

Exemple 2

Cet exemple combine des filtrages de sites et de liens externes :

```
TABLEAU ALPHA(0;$filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*www.google.*") `Sélectionner "google"
AJOUTER A TABLEAU($AllowDeny;Faux) `Interdire ce lien
WA FIXER FILTRES URL(MaZoneW;$filters;$AllowDeny)

TABLEAU ALPHA(0;$filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*www.google.*") `Sélectionner "google"
AJOUTER A TABLEAU($AllowDeny;Faux)
 //Faux : ce lien devrait être ouvert dans un navigateur externe, mais ce paramétrage est sans effet car le lien
sera bloqué
 //du fait du filtrage d'URL.
WA FIXER FILTRES LIENS EXTERNES(MaZoneW;$filters;$AllowDeny)
```

WA FIXER FILTRES URL

WA FIXER FILTRES URL ({* ;} objet ; tabFiltres ; tabAutorisRefus)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabFiltres	Tableau chaîne	⇒	Tableau de filtres
tabAutorisRefus	Tableau booléen	⇒	Tableau autoriser-refuser

Description

La commande **WA FIXER FILTRES URL** permet de mettre en place un ou plusieurs filtre(s) pour la zone Web désignée par les paramètres * et *objet*.

Avant le chargement de toute page demandée par l'utilisateur, 4D consulte la liste des filtres afin de vérifier si l'URL cible est autorisé ou non. L'évaluation de l'URL est basée sur le contenu des tableaux *tabFiltres* et *tabAutorisRefus* s'ils ont été définis.

Si l'URL demandé n'est pas autorisé, il n'est pas chargé et l'événement formulaire Sur filtrage URL est généré (cf. commande **Evenement formulaire**).

Les tableaux *tabFiltres* et *tabAutorisRefus* doivent être synchronisés.

- Chaque ligne du tableau *tabFiltres* doit contenir un URL devant être filtré. Vous pouvez utiliser le * comme joker de remplacement pour un ou plusieurs caractère(s).
- Chaque ligne correspondante dans le tableau *tabAutorisRefus* doit contenur un booléen indiquant si l'URL doit être autorisé (**Vrai**) ou refusé (**Faux**).

En cas de contradiction au niveau des paramétrages (autorisation et refus d'un même URL), le paramétrage pris en compte est le dernier.

Pour désactiver le filtrage des URLs, appelez la commande en lui passant des tableaux vides ou en passant respectivement les valeurs "" et **Vrai** dans la dernière ligne des tableaux *tabFiltres* et *tabAutorisRefus*.

Une fois la commande exécutée, les filtres deviennent une propriété de la zone Web. Si les tableaux *tabFiltres* et *tabAutorisRefus* sont supprimés ou réinitialisés, les filtres restent actifs tant que la commande n'a pas été exécutée à nouveau. Pour connaître les filtres actifs pour une zone, vous devez utiliser la commande **WA LIRE FILTRES URL**.

Important : Le filtrage des URLs effectué par cette commande s'applique uniquement à la variable "URL" associée à la zone Web (variable généralement saisissable et affichée dans le formulaire).
Le filtrage ne s'applique pas à la commande **WA OUVRIR URL** ni aux autres commandes de navigation.

Exemple 1

Vous souhaitez interdire l'accès à tous les sites web .org, .net et .fr :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*.org")
AJOUTER A TABLEAU($AllowDeny;Faux)
AJOUTER A TABLEAU($filters;"*.net")
AJOUTER A TABLEAU($AllowDeny;Faux)
AJOUTER A TABLEAU($filters;"*.fr")
AJOUTER A TABLEAU($AllowDeny;Faux)
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

Exemple 2

Vous souhaitez interdire l'accès à tous les sites web sauf les sites russes (.ru) :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*") `Tout sélectionner
AJOUTER A TABLEAU($AllowDeny;Faux) `Tout interdire
AJOUTER A TABLEAU($filters;"www.*.ru") `Sélectionner *.ru
AJOUTER A TABLEAU($AllowDeny;Vrai) `Autoriser
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

Exemple 3

Vous souhaitez donner accès aux sites Web 4D uniquement (.com, .fr, .es, etc.) :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*") `Tout sélectionner
AJOUTER A TABLEAU($AllowDeny;Faux) `Tout interdire
AJOUTER A TABLEAU($filters;"www.4D.*") `Sélectionner 4d.fr, 4d.com...
AJOUTER A TABLEAU($AllowDeny;Vrai) `Autoriser
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

Exemple 4

Vous souhaitez autoriser l'accès local à la documentation uniquement (située dans le dossier C://doc) :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*") `Tout sélectionner
AJOUTER A TABLEAU($AllowDeny;Faux) `Tout interdire
AJOUTER A TABLEAU($filters;"file://C:/doc/*") `Sélectionner le chemin file:// autorisé
AJOUTER A TABLEAU($AllowDeny;Vrai)->Autoriser
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

Exemple 5

Vous souhaitez autoriser tous les sites sauf un, par exemple celui d'Elcaro :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*")
AJOUTER A TABLEAU($AllowDeny;Vrai) `Tout autoriser
AJOUTER A TABLEAU($filters;"*elcaro*") `Interdire tout ce qui contient elcaro
AJOUTER A TABLEAU($AllowDeny;Faux)
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

Exemple 6

Vous souhaitez interdire des adresses IP spécifiques :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*") `Tout sélectionner
AJOUTER A TABLEAU($AllowDeny;Vrai) `Tout autoriser
AJOUTER A TABLEAU($filters;"86.83.*") `Sélectionner les IP débutant par 86.83.
AJOUTER A TABLEAU($AllowDeny;Faux) `Interdire
AJOUTER A TABLEAU($filters;"86.1*") `Sélectionner les IP débutant par 86.1 (86.10, 86.135 etc.)
AJOUTER A TABLEAU($AllowDeny;Faux) `Interdire
 `A noter que l'adresse IP d'un domaine peut varier
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

WA FIXER PREFERENCE

WA FIXER PREFERENCE ({* ;} objet ; sélecteur ; valeur)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
sélecteur	Entier long	⇒	Préférence à modifier
valeur	Booléen	⇒	Valeur de la préférence (Vrai = autorisé, Faux = non autorisé)

Description

La commande **WA FIXER PREFERENCE** permet de fixer différentes préférences pour la zone Web désignée par les paramètres * et *objet*.

Passez dans le paramètre *sélecteur* la préférence à modifier et dans *valeur* la valeur à lui attribuer. Vous pouvez passer dans *sélecteur* l'une des constantes suivantes, placées dans le thème "**Zone Web**" :

Constante	Type	Valeur	Comment
WA autoriser applets Java	Entier long	1	Autoriser l'exécution d'applets Java dans la zone Web
WA autoriser déposer URL	Entier long	101	Autoriser le déposer d'URLs ou de fichiers dans la zone Web (défaut = Faux)
WA autoriser inspecteur Web	Entier long	100	Autoriser l'affichage de l'inspecteur Web dans la zone
WA autoriser JavaScript	Entier long	2	Autoriser l'exécution de code JavaScript dans la zone Web
WA autoriser menu contextuel	Entier long	4	Autoriser l'affichage du menu contextuel standard dans la zone Web
WA autoriser plugins	Entier long	3	Autoriser l'installation de plug-ins dans la zone Web

Pour chaque préférence, passez **Vrai** dans *valeur* pour l'activer et **Faux** pour l'inactiver.

Note : L'usage de plugins Web et d'applets Java dans les zones Web **est déconseillé** car ils peuvent entraîner des instabilités dans le fonctionnement de 4D, notamment au niveau de la gestion des événements.

Exemple

Vous souhaitez autoriser le déposer d'URLs dans la zone Web 'myarea' :

```
WA FIXER PREFERENCE(*;"myarea";WA autoriser déposer URL;Vrai)
```

WA Lire contenu page

WA Lire contenu page ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Chaîne	⇐	Code HTML source

Description

La commande **WA Lire contenu page** retourne le code HTML de la page courante ou en cours d'affichage dans la zone Web désignée par les paramètres * et *objet*.

Cette commande retourne une chaîne vide si le contenu de la page courante n'est pas disponible.

WA Lire dernier URL filtre

WA Lire dernier URL filtre ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Chaîne	⇐	Dernier URL filtré

Description

La commande **WA Lire dernier URL filtre** retourne le dernier URL ayant été filtré dans la zone Web désignée par les paramètres * et *objet*.
L'URL peut avoir été filtré pour l'une des raisons suivantes :

- l'URL est interdit à cause d'un filtre (commande **WA FIXER FILTRES URL**),
- le lien est ouvert dans le navigateur par défaut (commande **WA FIXER FILTRES LIENS EXTERNES**),
- l'URL tentait d'ouvrir une fenêtre pop up.

Il est judicieux d'appeler cette commande dans le contexte des événements formulaire Sur filtrage URL, Sur ouverture lien externe et Sur refus ouverture fenêtre afin de connaître l'URL filtré. Pour plus d'informations, reportez-vous à la description de la commande **Evenement formulaire**.

WA LIRE DERNIERE ERREUR URL

WA LIRE DERNIERE ERREUR URL ({* ;} objet ; url ; description ; codeErreur)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
url	Chaîne	⇐	URL à l'origine de l'erreur
description	Chaîne	⇐	Description de l'erreur (Mac OS)
codeErreur	Entier long	⇐	Code d'erreur

Description

La commande **WA LIRE DERNIERE ERREUR URL** vous permet de récupérer plusieurs informations relatives à la dernière erreur ayant eu lieu dans la zone Web désignée par les paramètres * et *objet*.
Ces informations sont retournées dans trois variables :

- *url* : l'URL ayant provoqué l'erreur.
- *description* (Mac OS uniquement) : un texte décrivant l'erreur (si disponible). S'il n'est pas possible d'associer un texte à l'erreur, une chaîne vide est retournée. Sous Windows, ce paramètre est toujours retourné vide.
- *codeErreur* : code de l'erreur.
  - Si le code est >=400, il s'agit d'une erreur liée au protocole HTTP. Pour plus d'informations sur ce type d'erreur, reportez-vous à l'adresse http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
  - Sinon, il s'agit d'une erreur retournée par le WebKit (Mac OS) ou ActiveX (Windows).

Il est judicieux d'appeler cette commande dans le cadre de l'événement formulaire Sur erreur chargement URL afin de connaître la cause de l'erreur qui vient de se produire. Pour plus d'informations, reportez-vous à la description de la commande **Evenement formulaire**.

WA LIRE FILTRES LIENS EXTERNES

WA LIRE FILTRES LIENS EXTERNES ({* ;} objet ; tabFiltres ; tabAutorisRefus)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabFiltres	Tableau chaîne	⇐	Tableau de filtres
tabAutorisRefus	Tableau booléen	⇐	Tableau autoriser-refuser

Description

La commande **WA LIRE FILTRES LIENS EXTERNES** retourne dans les tableaux *tabFiltres* et *tabAutorisRefus* les filtres de liens externes de la zone Web désignée par les paramètres * et *objet*. Si aucun filtre n'est actif, les tableaux sont retournés vides.

Les filtres sont installés par la commande **WA FIXER FILTRES LIENS EXTERNES**. Si les tableaux ont été réinitialisés au cours de la session, la commande **WA LIRE FILTRES LIENS EXTERNES** vous permet de connaître le paramétrage courant.

WA LIRE FILTRES URL

WA LIRE FILTRES URL ({* ;} objet ; tabFiltres ; tabAutorisRefus)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabFiltres	Tableau chaîne	⇐	Tableau de filtres
tabAutorisRefus	Tableau booléen	⇐	Tableau autoriser-refuser

Description

La commande **WA LIRE FILTRES URL** retourne dans les tableaux *tabFiltres* et *tabAutorisRefus* les filtres actifs dans la zone Web désignée par les paramètres * et *objet*. Si aucun filtre n'est actif, les tableaux sont retournés vides.

Les filtres sont installés par la commande **WA FIXER FILTRES URL**. Si les tableaux ont été réinitialisés au cours de la session, la commande **WA LIRE FILTRES URL** vous permet de connaître le paramétrage courant.

WA LIRE HISTORIQUE URL

WA LIRE HISTORIQUE URL ({* ;} objet ; tabsUrls {; sens {; tabTitres}})

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabsUrls	Tableau chaîne	⇐	Tableau des URLs visités
sens	Entier long	⇒	0 ou omis=Liste des URLs précédents, 1=Liste des URLs suivants
tabTitres	Tableau chaîne	⇐	Tableau des titres de fenêtres

Description

La commande **WA LIRE HISTORIQUE URL** retourne un ou deux tableaux contenant les URLs visités au cours de la session dans la zone Web désignée par les paramètres * et *objet*. Elle permet de construire une interface de navigation personnalisée.

Les informations fournies concernent la session, c'est-à-dire la navigation effectuée dans une même zone Web tant que le formulaire n'a pas été refermé.

Le tableau *tabUrls* est rempli avec la liste des URLs visités. En fonction de la valeur du paramètre *direction* (s'il est passé), le tableau récupère la liste des URLs précédents (fonctionnement par défaut) ou la liste des URLs suivants. Ces listes correspondent au contenu des boutons standard Précédent et Suivant des navigateurs.

Les URLs sont classés par ordre chronologique.

Passez dans *direction* une valeur indiquant la liste à récupérer. Vous pouvez utiliser l'une des constantes suivantes, placées dans le thème "**Zone Web**" :

Constante	Type	Valeur
WA URLs précédents	Entier long	0
WA URLs suivants	Entier long	1

Si vous omettez le paramètre *direction*, la valeur 0 est utilisée.

S'il est passé, le paramètre *tabTitres* contient la liste des noms de fenêtres associés aux URLs. Ce tableau est synchronisé avec le tableau *tabUrls*.

WA LIRE PREFERENCE

WA LIRE PREFERENCE ({* ;} objet ; sélecteur ; valeur)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
sélecteur	Entier long	⇒	Préférence à lire
valeur	Variable	⇐	Valeur courante de la préférence

Description

La commande **WA LIRE PREFERENCE** permet de lire la valeur courante d'une préférence dans la zone Web désignée par les paramètres * et *objet*.

Passez dans le paramètre *sélecteur* la préférence à lire. Vous pouvez passer l'une des constantes suivantes, placées dans le thème "**Zone Web**" :

Constante	Type	Valeur	Comment
WA autoriser applets Java	Entier long	1	Autoriser l'exécution d'applets Java dans la zone Web
WA autoriser déposer URL	Entier long	101	Autoriser le déposer d'URLs ou de fichiers dans la zone Web (défaut = Faux)
WA autoriser inspecteur Web	Entier long	100	Autoriser l'affichage de l'inspecteur Web dans la zone
WA autoriser JavaScript	Entier long	2	Autoriser l'exécution de code JavaScript dans la zone Web
WA autoriser menu contextuel	Entier long	4	Autoriser l'affichage du menu contextuel standard dans la zone Web
WA autoriser plugins	Entier long	3	Autoriser l'installation de plug-ins dans la zone Web

Passez dans le paramètre *valeur* une variable devant recevoir la valeur courante de la préférence. Le type de la variable dépend de la préférence. La variable *valeur* est toujours de type booléen : elle contient **Vrai** si la préférence est active et **Faux** sinon.

WA Lire titre page

WA Lire titre page ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Chaîne	⇐	Titre de la page courante

Description

La commande **WA Lire titre page** retourne le titre de la page courante ou en cours d'affichage dans la zone Web désignée par les paramètres * et *objet*. Le titre correspond à la balise HTML "Title".

Cette commande retourne une chaîne vide s'il n'y a pas de titre disponible à l'URL courant.

WA Lire URL courant

WA Lire URL courant ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Chaîne	⇐	URL actuellement chargé dans la zone Web

Description

La commande **WA Lire URL courant** retourne l'adresse URL de la page affichée dans la zone Web désignée par les paramètres * et *objet*.

Si l'URL courant n'est pas disponible, la commande retourne une chaîne vide.

Si la page Web est entièrement chargée, la valeur retournée par la fonction est identique à celle de la variable "URL" associée à la zone Web. Si la page est en cours de chargement, les deux valeurs seront différentes : la fonction retourne l'URL entièrement chargé et la variable contient l'URL en cours de chargement.

Exemple

La page affichée est l'URL "www.apple.com" et la page "www.4d.com" est en cours de chargement :

```
$url:=WA Lire URL courant(MaZoneW)  `retourne "http://www.apple.com"
 `La variable URL associée contient "http://www.4d.com"
```

WA OUVRIR URL

WA OUVRIR URL ({* ;} objet ; url)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
url	Chaîne	⇒	URL à charger dans la zone Web

Description

La commande **WA OUVRIR URL** charge dans la zone Web désignée par les paramètres *** et *objet* l'URL passé dans le paramètre *url*.

Si une chaîne vide est passée dans *url*, la commande **WA OUVRIR URL** ne fait rien et aucune erreur n'est générée. Pour charger une page vide dans la zone Web, passez la chaîne "about:blank" dans *url*.

Comme la commande **OUVRIR URL**, **WA OUVRIR URL** accepte plusieurs types de syntaxes dans le paramètre *url* pour désigner les fichiers :

- syntaxe posix : "file:///c:/Mon%20Fichier"
- syntaxe système : "c:\MonDossier\MonFichier" (Windows) ou "MonDisque:MonDossier:MonFichier" (Mac OS).

Note : Par compatibilité, la syntaxe "file://" (utilisation de deux "/") est acceptée dans 4D mais elle n'est pas conforme aux RFC. Il est conseillé d'utiliser la syntaxe "file:///" (trois "/") qui est conforme aux RFC.

Sous Mac OS, quand FileVault est activé, vous devez utiliser la syntaxe posix. Vous pouvez transformer les chemins système via la commande **Convertir chemin systeme vers POSIX**.

Cette commande a le même effet que la modification de la valeur de la variable "URL" associée à la zone. Par exemple, si la variable de la zone est nommée MaZoneW_url :

```
MaZoneW_url:="http://www.4d.com/"
```

équivaut à :

```
WA OUVRIR URL(MaZoneW;"http://www.4d.com/")
```

WA OUVRIR URL PRECEDENT

WA OUVRIR URL PRECEDENT ({* ;} objet)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande **WA OUVRIR URL PRECEDENT** charge dans la zone Web désignée par les paramètres * et *objet* l'URL précédent dans la séquence d'URLs ouverts.

S'il n'y a pas d'URL précédent, la commande ne fait rien. Vous pouvez tester la disponibilité d'un URL précédent à l'aide de la commande **WA URL precedent disponible**.

WA OUVRIR URL SUIVANT

WA OUVRIR URL SUIVANT ({* ;} objet)

Paramètre	Type		Description
*	Opérateur	⇨	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇨	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande **WA OUVRIR URL SUIVANT** charge dans la zone Web désignée par les paramètres *** et *objet* l'URL suivant dans la séquence d'URLs ouverts.

S'il n'y a pas d'URL suivant (c'est-à-dire si l'utilisateur n'a jamais effectué de retour à l'URL précédent), la commande ne fait rien. Vous pouvez tester la disponibilité d'un URL suivant à l'aide de la commande **WA URL suivant disponible**.

WA REDUIRE TEXTE PAGE

WA REDUIRE TEXTE PAGE ({* ;} objet)

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande **WA REDUIRE TEXTE PAGE** réduit la taille du texte affiché dans la zone Web désignée par les paramètres *** et *objet*.
Sous Mac OS, la portée de cette commande est la session 4D : le paramétrage effectué n'est pas conservé après la fermeture de l'application 4D.
Sous Windows, la portée de cette commande est globale : le paramétrage est conservé après la fermeture de l'application 4D.

WA URL precedent disponible

WA URL precedent disponible ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Booléen	⇐	Vrai s'il existe un URL précédent dans la séquence d'URLs ouverts, Faux sinon

Description

La commande **WA URL precedent disponible** permet de savoir s'il existe un URL précédent disponible dans la séquence d'URLs ouverts dans la zone Web désignée par les paramètres * et *objet*.

La commande retourne **Vrai** si un URL existe et **Faux** sinon. Cette commande permet notamment, dans la cadre d'une interface personnalisée, d'activer ou d'inactiver des boutons de navigation.

WA URL suivant disponible

WA URL suivant disponible ({* ;} objet) -> Résultat

Paramètre	Type		Description
*	Opérateur	⇒	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	⇒	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Booléen	⇐	Vrai s'il existe un URL suivant dans la séquence d'URLs ouverts, Faux sinon

Description

La commande **WA URL suivant disponible** permet de savoir s'il existe un URL suivant disponible dans la séquence d'URLs ouverts dans la zone Web désignée par les paramètres * et *objet*.

La commande retourne **Vrai** si un URL existe et **Faux** sinon. Cette commande permet notamment, dans la cadre d'une interface personnalisée, d'activer ou d'inactiver des boutons de navigation.

Liste des thèmes de constantes

- 4D Write Pro
- Accès objets développement
- Attributs de texte multistyle
- BLOB
- Caractères latins ISO
- Chercher fenetre
- Client HTTP
- Codes ASCII
- Communications
- Compression images
- Conteneur de données
- Couleurs
- Creer fenetre
- Creer fenetre formulaire
- Dictionnaires
- Documents système
- Dossier Système
- Environnement 4D
- Euro monnaies
- Evénements (Modifiers)
- Evénements (types)
- Evénements de la base
- Evénements formulaire
- Evénements trigger
- Expressions
- Fenêtre
- FIXER COULEUR RVB
- Fonctions mathématiques
- Formatages système
- Formats d'affichage des dates
- Formats d'affichage des heures
- Formats d'affichage des images
- Interfaces de plate-forme
- Journal d'événements
- Jours et mois
- LDAP
- Licence disponible
- Liens
- List box
- List box pied calcul
- Listes hiérarchiques
- Maintenance fichier de données
- Moteur de la base
- Noms des métadonnées images
- Numéros de port TCP
- Objets de formulaire (Accès)
- Objets de formulaire (Propriétés)
- Options d'impression
- Paramètre des graphes
- Paramètres de formulaire
- Paramètres de la base
- PHP
- PROFONDEUR ECRAN
- Propriétés des lignes de menu
- Propriétés des ressources
- Propriétés plate-forme
- QR Commandes
- QR Destination de sortie
- QR Encadrements
- QR Lignes pour Propriétés
- QR Opérateurs
- QR Propriétés de document
- QR Propriétés de texte
- QR Propriétés de zone
- QR Types d'états
- Recherches
- Sauvegarde et restitution
- Serveur Web
- Signatures système standard
- SQL
- Statut du process
- Styles de caractères
- Texte multistyle
- Touches de fonction

- Touches équivalents clavier
- Transformation des images
- Type de liste des polices
- Type digest
- Type du process
- Type index
- Types champs et variables
- Types objets formulaire
- Valeurs des métadonnées images
- Valeurs pour Actions standard associée
- Valeurs Texte pour Action standard associée
- Web Services (Client)
- Web Services (Serveur)
- XML
- Zone de formulaire
- Zone Web

Constante	Type	Valeur	Comment
wk 4wp	Entier long	4	Le document 4D Write Pro est sauvegardé dans un format d'archive natif (HTML *zippé* avec images stockées dans un dossier séparé). Les expressions 4D ne sont pas calculées et les balises 4D spécifiques sont incluses. Ce format est particulièrement approprié pour la sauvegarde et l'archivage de documents 4D Write Pro sur disque sans aucune perte d'informations.
wk armenian	Entier long	19	
wk author	Chaîne	author	
wk auto	Entier long	0	
wk background clip	Chaîne	backgroundClip	
wk background color	Chaîne	backgroundColor	
wk background image	Chaîne	backgroundImage	
wk background origin	Chaîne	backgroundOrigin	
wk background position h	Chaîne	backgroundPositionH	
wk background position v	Chaîne	backgroundPositionV	
wk background repeat	Chaîne	backgroundRepeat	
wk background size h	Chaîne	backgroundSizeH	
wk background size v	Chaîne	backgroundSizeV	
wk bar	Entier long	4	
wk baseline	Entier long	4	
wk border box	Entier long	0	
wk border color	Chaîne	borderColor	
wk border color bottom	Chaîne	borderColorBottom	
wk border color left	Chaîne	borderColorLeft	
wk border color right	Chaîne	borderColorRight	
wk border color top	Chaîne	borderColorTop	
wk border radius	Chaîne	borderRadius	
wk border style	Chaîne	borderStyle	
wk border style bottom	Chaîne	borderStyleBottom	
wk border style left	Chaîne	borderStyleLeft	
wk border style right	Chaîne	borderStyleRight	
wk border style top	Chaîne	borderStyleTop	
wk border width	Chaîne	borderWidth	
wk border width bottom	Chaîne	borderWidthBottom	
wk border width left	Chaîne	borderWidthLeft	

Constante	Type	Valeur	Comment
wk border width right	Chaîne	borderWidthRight	
wk border width top	Chaîne	borderWidthTop	
wk bottom	Entier long	1	
wk capitalize	Entier long	1	
wk center	Entier long	2	
wk circle	Entier long	11	
wk cjk ideographic	Entier long	24	
wk club	Entier long	27	
wk company	Chaîne	company	
wk contain	Entier long	-1	
wk content box	Entier long	2	
wk cover	Entier long	-2	
wk custom	Entier long	29	
wk dashed	Entier long	3	
wk date creation	Chaîne	dateCreation	
wk date modified	Chaîne	dateModified	
wk decimal	Entier long	3	
wk decimal greek	Entier long	28	
wk decimal leading zero	Entier long	13	
wk default	Entier long	-1	
wk diamond	Entier long	26	
wk direction	Chaîne	direction	
wk disc	Entier long	10	
wk dotted	Entier long	2	
wk double	Entier long	4	
wk dpi	Chaîne	dpi	
wk end text	Entier long	0	
wk false	Entier long	0	
wk font	Chaîne	font	
wk font bold	Chaîne	fontBold	
wk font family	Chaîne	fontFamily	
wk font italic	Chaîne	fontItalic	
wk font size	Chaîne	fontSize	
wk georgian	Entier long	20	
wk groove	Entier long	6	
wk hebrew	Entier long	21	
wk height	Chaîne	height	
wk hidden	Entier long	5	
wk hiragana	Entier long	22	
wk hollow square	Entier long	25	
wk html	Entier	1	Code HTML formaté ("pretty print"), facilitant le débogage

Constante	Type	Valeur	Comment
wk image	Chaîne	image	
wk image alternative text	Chaîne	imageAltText	
wk inset	Entier long	8	
wk inside	Chaîne	Inside	
wk justify	Entier long	5	
wk katakana	Entier long	23	
wk layout unit	Chaîne	userUnit	
wk left	Entier long	0	
wk left to right	Entier long	0	
wk line height	Chaîne	lineHeight	
wk linethrough	Entier long	2	
wk list auto	Entier long	2147483647	
wk list font	Chaîne	listFont	
wk list font family	Chaîne	listFontFamily	
wk list start number	Chaîne	listStartNumber	
wk list string format LTR	Chaîne	listStringFormatLtr	
wk list string format RTL	Chaîne	listStringFormatRtl	
wk list style image	Chaîne	listStyleImage	
wk list style image height	Chaîne	listStyleImageHeight	
wk list style type	Chaîne	listStyleType	
wk lower greek	Entier long	18	
wk lower latin	Entier long	14	
wk lower roman	Entier long	15	
wk lowercase	Entier long	2	
wk margin	Chaîne	margin	
wk margin bottom	Chaîne	marginBottom	
wk margin left	Chaîne	marginLeft	
wk margin right	Chaîne	marginRight	
wk margin top	Chaîne	marginTop	
wk middle	Entier long	2	
wk mime html	Entier long	1	Le document 4D Write Pro est sauvegardé au format MIME HTML avec les documents html et les images embarqués en tant que parties MIME (encodées en base64). Les expressions sont calculées et les balises 4D spécifiques sont supprimées. Ce format est particulièrement adapté à l'envoi de mails au format HTML à l'aide de la commande **SMTP_QuickSend**.
wk min height	Chaîne	minHeight	
wk min width	Chaîne	minWidth	
wk mixed	Entier long	-2147483648	
wk new line style sheet	Chaîne	newLineStyleSheet	
wk no repeat	Entier long	3	
wk none	Entier long	0	

Constante	Type	Valeur	Comment
wk normal	Entier long	0	Code HTML standard
wk notes	Chaîne	notes	
wk outset	Entier long	9	
wk outside	Chaîne	Outside	
wk padding	Chaîne	padding	
wk padding bottom	Chaîne	paddingBottom	
wk padding box	Entier long	1	
wk padding left	Chaîne	paddingLeft	
wk padding right	Chaîne	paddingRight	
wk padding top	Chaîne	paddingTop	
wk page web complète	Entier long	2	Extension .htm ou .html. Le document est sauvegardé au format HTML standard et ses ressources sont sauvegardées séparément. Les balises 4D spécifiques sont supprimées et les expressions sont calculées. Ce format est particulièrement adapté à l'affichage d'un document 4D Write Pro dans un navigateur Web.
wk page web html 4D	Entier long	3	Le document 4D Write Pro est sauvegardé au format HTML et inclut les balises 4D spécifiques ; chaque expression est insérée sous forme d'espace insécable. Comme ce format est sans perte, il est approprié pour le stockage dans un champ texte.
wk range end	Chaîne	rangeEnd	
wk range owner	Chaîne	rangeOwner	
wk range start	Chaîne	rangeStart	
wk repeat	Entier long	0	
wk repeat x	Entier long	1	
wk repeat y	Entier long	2	
wk ridge	Entier long	7	
wk right	Entier long	1	
wk right to left	Entier long	1	
wk semi transparent	Entier long	5	
wk small uppercase	Entier long	4	
wk solid	Entier long	1	
wk square	Entier long	12	
wk start text	Entier long	1	
wk style sheet	Chaîne	styleSheet	
wk subject	Chaîne	subject	
wk subscript	Entier long	6	
wk superscript	Entier long	5	
wk tab stop offsets	Chaîne	tabStopOffsets	
wk tab stop types	Chaîne	tabStopTypes	
wk text align	Chaîne	textAlign	
wk text color	Chaîne	color	
wk text indent	Chaîne	textIndent	
wk text linethrough color	Chaîne	textLinethroughColor	
wk text linethrough style	Chaîne	textLinethroughStyle	
wk text shadow color	Chaîne	textShadowColor	

Constante	Type	Valeur	Comment
wk text shadow offset	Chaîne	textShadowOffset	
wk text transform	Chaîne	textTransform	
wk text underline color	Chaîne	textUnderlineColor	
wk text underline style	Chaîne	textUnderlineStyle	
wk title	Chaîne	title	
wk top	Entier long	0	
wk transparent	Entier long	-1	
wk true	Entier long	1	
wk underline	Entier long	1	
wk unit cm	Chaîne	cm	
wk unit inch	Chaîne	in	
wk unit mm	Chaîne	mm	
wk unit percent	Chaîne	%	
wk unit pt	Chaîne	pt	
wk unit px	Chaîne	px	
wk upper latin	Entier long	16	
wk upper roman	Entier long	17	
wk uppercase	Entier long	3	
wk value unit not percentage	Entier long	-100000	
wk value unit percentage	Entier long	-100001	
wk version	Chaîne	version	
wk vertical align	Chaîne	verticalAlign	
wk width	Chaîne	width	
wk word	Entier long	6	

Les constantes de 4D Write Pro sont détaillées dans le manuel de 4D Write Pro. Pour plus d'informations, veuillez vous référer au paragraphe **4D Write Pro - Langage**.

Constante	Type	Valeur	Comment

Accès objets développement

Constante	Type	Valeur	Comment
Attribut exécutée sur serveur	Entier long	8	Correspond à l'option "Exécuter sur serveur"
Attribut invisible	Entier long	1	Correspond à l'option "Invisible"
Attribut nom dossier	Entier long	1024	Nom de dossier pour la méthode (attribut "folder"). Lorsque vous passez cette constante, vous devez passer un nom de dossier dans *valeurAttribut* : • si le nom correspond à un dossier valide, la méthode sera placée dans ce dossier parent, • si le dossier n'existe pas, la commande ne change rien au niveau du dossier parent, • si vous passez une chaîne vide, la méthode sera placée au niveau racine.
Attribut partagée	Entier long	5	Correspond à l'option "Partagée entre composants et base hôte"
Attribut publiée SOAP	Entier long	3	Correspond à l'option "Offerte comme Web Service"
Attribut publiée SQL	Entier long	7	Correspond à l'option "Disponible via SQL"
Attribut publiée Web	Entier long	2	Correspond à l'option "Disponible via les balises HTML et les URLs 4D (4DACTION...)"
Attribut publiée WSDL	Entier long	4	Correspond à l'option "Publiée dans WSDL". N'est prise en compte que si l'option "Offerte comme Web Service" est activée.
Chemin formulaire projet	Entier long	4	Chemin des méthodes formulaire projet et de toutes leurs méthodes objet. Exemples : *[projectForm]/monForm/{formMethod}* *[projectForm]/monForm/bouton1* *[projectForm]/monForm/ma%2liste* *[projectForm]/monForm2/bouton1*
Chemin formulaire table	Entier long	16	Chemin des méthodes formulaire table et de toutes leurs méthodes objet. Exemples : *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/bouton1* *[tableForm]/table_1/Form1/ma%2liste* *[tableForm]/table_2/Form1/ma%2liste*
Chemin méthode base	Entier long	2	Chemin des méthodes base définies (nom anglais). Liste de ces méthodes : *[databaseMethod]/onStartup* *[databaseMethod]/onExit* *[databaseMethod]/onDrop* *[databaseMethod]/onBackupStartup* *[databaseMethod]/onBackupShutdown* *[databaseMethod]/onWebConnection* *[databaseMethod]/onWebAuthentication* *[databaseMethod]/onWebSessionSuspend* *[databaseMethod]/onServerStartup* *[databaseMethod]/onServerShutdown* *[databaseMethod]/onServerOpenConnexion* *[databaseMethod]/onServerCloseConnection* *[databaseMethod]/onSystemEvent* *[databaseMethod]/onSqlAuthentication*
Chemin Méthode projet	Entier long	1	Nom de la méthode. Exemple : *MaMethodeProjet*
Chemin tous les objets	Entier long	31	Combinaison des chemins de toutes les méthodes de la base
Chemin trigger	Entier long	8	Chemin des triggers de la base. Exemples : *[trigger]/table_1* *[trigger]/table_2*
Code avec tokens	Entier long	1	Inclure les tokens dans le code exporté
Sur objet verrouillé abandonner	Entier long	0	Le chargement de l'objet est abandonné (fonctionnement par défaut)
Sur objet verrouillé confirmer	Entier long	2	4D affiche une boîte de dialogue vous permettant de choisir de réessayer ou d'abandonner. En mode distant, cette option n'est pas prise en charge (le chargement est abandonné)
Sur objet verrouillé réessayer	Entier long	1	4D tente de charger l'objet jusqu'à ce qu'il soit libéré

Attributs de texte multistyle

Constante	Type	Valeur	Comment
Attribut couleur fond	Entier long	8	*valeurAttribut*=Valeur hexadécimale ou nom de couleur HTML (Windows uniquement)
Attribut couleur texte	Entier long	7	*valeurAttribut*=Valeur hexadécimale ou nom de couleur HTML
Attribut nom de police	Entier long	5	*valeurAttribut*=Nom de la famille de police (chaîne)
Attribut style barré	Entier long	3	*valeurAttribut*=0 : pas d'attribut barré pour la sélection *valeurAttribut*=1 : attribut barré pour la sélection
Attribut style gras	Entier long	1	*valeurAttribut*=0 : pas d'attribut gras pour la sélection *valeurAttribut*=1 : attribut gras pour la sélection
Attribut style italique	Entier long	2	*valeurAttribut*=0 : pas d'attribut italique pour la sélection *valeurAttribut*=1 : attribut italique pour la sélection
Attribut style souligné	Entier long	4	*valeurAttribut*=0 : pas d'attribut souligné pour la sélection *valeurAttribut*=1 : attribut souligné pour la sélection
Attribut taille texte	Entier long	6	*valeurAttribut*=Nombre de points (numérique)

BLOB

Constante	Type	Valeur	Comment
Format réel double Macintosh	Entier long	2	
Format réel double PC	Entier long	3	
Format réel étendu	Entier long	1	
Format réel natif	Entier long	0	
GZIP méthode de compression compacte	Entier long	-1	Compression GZIP la plus compacte (au détriment de la vitesse à laquelle la compression et la décompression sont effectuées)
GZIP méthode de compression rapide	Entier long	-2	Compression/décompression GZIP la plus rapide (au détriment du taux de compression)
Mac chaîne en C	Entier long	0	
Mac chaîne pascal	Entier long	1	
Mac texte avec longueur	Entier long	2	
Mac texte sans longueur	Entier long	3	
Méthode de compression compacte	Entier long	1	Compression interne la plus compacte (au détriment de la vitesse à laquelle la compression et la décompression sont effectuées). Méthode par défaut.
Méthode de compression rapide	Entier long	2	Compression/décompression interne la plus rapide au détriment du taux de compression (une fois compressé, le BLOB prend plus de place)
Non compressé	Entier long	0	Pas de compression
Ordre octets Macintosh	Entier long	1	
Ordre octets natif	Entier long	0	
Ordre octets PC	Entier long	2	
UTF8 chaîne en C	Entier long	4	
UTF8 texte avec longueur	Entier long	5	
UTF8 texte sans longueur	Entier long	6	

Caractères latins ISO

Constante	Type	Valeur	Comment
ISO L1 0 majus circonflexe	Chaîne	Ô	
ISO L1 a aigu	Chaîne	á	
ISO L1 a circonflexe	Chaîne	â	
ISO L1 a grave	Chaîne	à	
ISO L1 A majus aigu	Chaîne	Á	
ISO L1 A majus circonflexe	Chaîne	Â	
ISO L1 A majus grave	Chaîne	À	
ISO L1 A majus tilde	Chaîne	Ã	
ISO L1 A majus umlaut	Chaîne	Ä	
ISO L1 A majus umlaut	Chaîne	Å	
ISO L1 a rond	Chaîne	å	
ISO L1 a tilde	Chaîne	ã	
ISO L1 a umlaut	Chaîne	ä	
ISO L1 ae ligature	Chaîne	æ	
ISO L1 AE majus ligature	Chaîne	&AELig;	
ISO L1 c cédille	Chaîne	ç	
ISO L1 C majus cédille	Chaîne	Ç	
ISO L1 Copyright	Chaîne	©	
ISO L1 e aigu	Chaîne	é	
ISO L1 e circonflexe	Chaîne	ê	
ISO L1 e grave	Chaîne	è	
ISO L1 E majus aigu	Chaîne	É	
ISO L1 E majus circonflexe	Chaîne	Ê	
ISO L1 E majus grave	Chaîne	È	
ISO L1 E majus unlaut	Chaîne	Ë	
ISO L1 e umlaut	Chaîne	ë	
ISO L1 Es zett allemand	Chaîne	ß	
ISO L1 Et commercial	Chaîne	&	
ISO L1 eth islandais	Chaîne	ð	
ISO L1 Eth majus islandais	Chaîne	Ð	
ISO L1 Guillemets	Chaîne	"	
ISO L1 i aigu	Chaîne	í	
ISO L1 i circonflexe	Chaîne	î	
ISO L1 i grave	Chaîne	ì	
ISO L1 I majus aigu	Chaîne	Í	
ISO L1 I majus circonflex	Chaîne	Î	
ISO L1 I majus grave	Chaîne	Ì	
ISO L1 I majus umlaut	Chaîne	Ï	
ISO L1 i umlaut	Chaîne	ï	
ISO L1 Inférieur	Chaîne	<	
ISO L1 Marque déposée	Chaîne	®	
ISO L1 N majus tilde	Chaîne	Ñ	
ISO L1 n tilde	Chaîne	ñ	
ISO L1 o aigu	Chaîne	ó	
ISO L1 o barré	Chaîne	ø	
ISO L1 o circonflexe	Chaîne	ô	
ISO L1 o grave	Chaîne	ò	
ISO L1 O majus aigu	Chaîne	Ó	
ISO L1 O majus barré	Chaîne	Ø	
ISO L1 O majus grave	Chaîne	Ò	
ISO L1 O majus tilde	Chaîne	Õ	
ISO L1 O majus umlaut	Chaîne	Ö	
ISO L1 o tilde	Chaîne	õ	
ISO L1 o umlaut	Chaîne	ö	
ISO L1 Supérieur	Chaîne	>	
ISO L1 thorn islandais	Chaîne	þ	
ISO L1 THORN majus islandais	Chaîne	Þ	
ISO L1 u aigu	Chaîne	ú	
ISO L1 u circonflexe	Chaîne	û	
ISO L1 u grave	Chaîne	ù	
ISO L1 U majus aigu	Chaîne	Ú	
ISO L1 U majus circonflexe	Chaîne	Û	
ISO L1 U majus grave	Chaîne	Ù	
ISO L1 U majus umlaut	Chaîne	Ü	
ISO L1 u umlaut	Chaîne	ü	
ISO L1 y aigu	Chaîne	ý	
ISO L1 Y majus aigu	Chaîne	Ý	
ISO L1 y umlaut	Chaîne	ÿ	

Chercher fenetre

Constante	Type	Valeur	Comment
_o_Dans zone contenu	Entier long	3	Plate-forme : Mac OS et Windows

Client HTTP

Constante	Type	Valeur	Comment
HTTP afficher dial auth	Entier long	4	*valeur* = 0 (ne pas afficher le dialogue) ou 1 (afficher le dialogue). Par défaut : 0 Cette option gère l'affichage de boîte de dialogue d'authentification lors de l'exécution de la commande **HTTP Get** ou **HTTP Request**. Par défaut, cette commande ne provoque jamais l'affichage de la boîte de dialogue, vous devez en principe utiliser la commande **HTTP AUTHENTIFIER**. Toutefois, si vous souhaitez qu'une boîte de dialogue d'authentification apparaisse pour que l'utilisateur saisisse ses identifiants, passez 1 dans valeur. La boîte de dialogue n'apparaît que si la requête requiert une authentification.
HTTP basic	Entier long	1	Utiliser la méthode d'authentification BASIC
HTTP compression	Entier long	6	*valeur* = 0 (ne pas compresser) ou 1 (compresser). Par défaut : 0 Cette option permet d'activer ou d'activer le mécanisme de compression des requêtes entre le client et le serveur, destiné à accélérer les échanges. Lorsque ce mécanisme est activé, le client HTTP utilise la compression deflate ou GZIP en fonction de la réponse du serveur.
HTTP digest	Entier long	2	Utiliser la méthode d'authentification DIGEST
HTTP effacer infos auth	Entier long	5	*valeur* = 0 (ne pas effacer les informations) ou 1 (les effacer). Par défaut : 0 Cette option permet d'indiquer à 4D de réinitialiser les informations d'authentification de l'utilisateur (nom d'utilisateur, mot de passe, méthode) après chaque exécution d'une commande **HTTP Get** ou **HTTP Request** dans un même process. Par défaut, ces informations sont conservées et réutilisées à chaque requête. Passez 1 dans *valeur* pour les effacer après chaque appel. A noter que quel que soit le paramétrage, les informations sont effacées lorsque le process est détruit.
HTTP méthode DELETE	Chaîne	DELETE	Voir la RFC 2616
HTTP méthode GET	Chaîne	GET	Voir la RFC 2616. Equivaut à utiliser la commande **HTTP Get**
HTTP méthode HEAD	Chaîne	HEAD	Voir la RFC 2616
HTTP méthode OPTIONS	Chaîne	OPTIONS	Voir la RFC 2616
HTTP méthode POST	Chaîne	POST	Voir la RFC 2616
HTTP méthode PUT	Chaîne	PUT	Voir la RFC 2616
HTTP méthode TRACE	Chaîne	TRACE	Voir la RFC 2616
HTTP redirections max	Entier long	3	*valeur* = nombre maximum de redirections acceptées Valeur par défaut = 2
HTTP suivre redirection	Entier long	2	*valeur* = 0 (ne pas accepter les redirections) ou 1 (accepter les redirections). Valeur par défaut = 1
HTTP timeout	Entier long	1	*valeur* = timeout de la requête cliente, exprimé en secondes. Le timeout est le délai d'attente du client HTTP en cas de non-réponse du serveur. A l'issue de ce délai, le client referme la session, la requête est perdue. Par défaut, ce délai est de 120 secondes. Il peut être modifié en raison de caractéristiques particulières (état du réseau, spécificités de la requête, etc.).

Codes ASCII

Constante	Type	Valeur	Comment
Apostrophe	Entier long	39	
Arobase	Entier long	64	
ASCII ACK	Entier long	6	
ASCII BEL	Entier long	7	
ASCII BS	Entier long	8	
ASCII CAN	Entier long	24	
ASCII CR	Entier long	13	
ASCII DC1	Entier long	17	
ASCII DC2	Entier long	18	
ASCII DC3	Entier long	19	
ASCII DC4	Entier long	20	
ASCII DEL	Entier long	127	
ASCII DLE	Entier long	16	
ASCII EM	Entier long	25	
ASCII ENQ	Entier long	5	
ASCII EOT	Entier long	4	
ASCII ESC	Entier long	27	
ASCII ETB	Entier long	23	
ASCII ETX	Entier long	3	
ASCII FF	Entier long	12	
ASCII FS	Entier long	28	
ASCII GS	Entier long	29	
ASCII HT	Entier long	9	
ASCII LF	Entier long	10	
ASCII NAK	Entier long	21	
ASCII NUL	Entier long	0	
ASCII RS	Entier long	30	
ASCII SI	Entier long	15	
ASCII SO	Entier long	14	
ASCII SOH	Entier long	1	
ASCII SP	Entier long	32	
ASCII STX	Entier long	2	
ASCII SUB	Entier long	26	
ASCII SYN	Entier long	22	
ASCII US	Entier long	31	
ASCII VT	Entier long	11	
Echappement	Entier long	27	
Effacement arrière	Entier long	8	
Entrée	Entier long	3	
Espace insécable	Entier long	202	
Espacement	Entier long	32	
Guillemets	Entier long	34	
Point	Entier long	46	
Retour à la ligne	Entier long	10	
Retour chariot	Entier long	13	
Tabulation	Entier long	9	

Communications

Constante	Type	Valeur	Comment
Bit de stop deux	Entier long	-16384	
Bit de stop un	Entier long	16384	
Bit de stop un et demi	Entier long	-32768	
Bits de données 5	Entier long	0	
Bits de données 6	Entier long	2048	
Bits de données 7	Entier long	1024	
Bits de données 8	Entier long	3072	
Parité impaire	Entier long	4096	
Parité paire	Entier long	12288	
Pas de parité	Entier long	0	
Port imprimante MacOS	Entier long	0	
Port série MacOS	Entier long	1	
Protocole aucun	Entier long	0	
Protocole DTR	Entier long	30	
Protocole XONXOFF	Entier long	20	
Vitesse 115200	Entier long	1022	
Vitesse 1200	Entier long	94	
Vitesse 1800	Entier long	62	
Vitesse 19200	Entier long	4	
Vitesse 230400	Entier long	1021	
Vitesse 2400	Entier long	46	
Vitesse 300	Entier long	380	
Vitesse 3600	Entier long	30	
Vitesse 4800	Entier long	22	
Vitesse 57600	Entier long	0	
Vitesse 600	Entier long	189	
Vitesse 7200	Entier long	14	
Vitesse 9600	Entier long	10	

Compression images

Constante	Type	Valeur	Comment

Conteneur de données

Constante	Type	Valeur	Comment
Données absentes conteneur	Entier long	-102	
Données image	Chaîne	PICT	
Données texte	Chaîne	TEXT	

Couleurs

Constante	Type	Valeur	Comment
Blanc	Entier long	0	
Bleu	Entier long	6	
Bleu clair	Entier long	7	
Bleu foncé	Entier long	5	
Gris	Entier long	14	
Gris clair	Entier long	12	
Gris foncé	Entier long	11	
Jaune	Entier long	1	
Marron	Entier long	13	
Marron foncé	Entier long	10	
Noir	Entier long	15	
Orange	Entier long	2	
Rouge	Entier long	3	
Vert	Entier long	8	
Vert foncé	Entier long	9	
Violet	Entier long	4	

Creer fenetre

Constante	Type	Valeur	Comment
_o_Avec bouton barre outils Mac	Entier long	8192	*** Constante obsolète ***
_o_Mode compositing	Entier long	4096	*** Constante obsolète ***
Aspect texture	Entier long	2048	
Avec barre de titre active	Entier long	1	
Avec case de contrôle de taille	Entier long	4	
Avec case de zoom	Entier long	8	
Avec mode plein écran Mac	Entier long	65536	
Avec titre de fenêtre	Entier long	2	
Dialogue modal	Entier long	1	
Dialogue modal déplaçable	Entier long	5	Utilisable en fenêtre flottante
Dialogue ombré	Entier long	3	Utilisable en fenêtre flottante
Dialogue simple	Entier long	2	Utilisable en fenêtre flottante
Fenêtre à coins arrondis	Entier long	16	
Fenêtre feuille	Entier long	33	
Fenêtre feuille redim	Entier long	34	
Fenêtre palette	Entier long	1984	Utilisable en fenêtre flottante
Fenêtre pop up	Entier long	32	
Fenêtre standard	Entier long	8	
Fenêtre standard de taille fixe	Entier long	4	
Fenêtre standard sans zoom	Entier long	0	

Creer fenetre formulaire

Constante	Type	Valeur	Comment
_o_Avec bouton barre outils Mac OS	Entier long	8192	*** Constante obsolète ***
_o_Form mode compositing	Entier long	4096	*** Constante obsolète ***
A droite	Entier long	196608	
A gauche	Entier long	131072	
Centrée horizontalement	Entier long	65536	
Centrée verticalement	Entier long	262144	
En bas	Entier long	393216	
En haut	Entier long	327680	
Form avec mode plein écran Mac	Entier long	65536	
Form dialogue modal	Entier long	1	
Form dialogue modal déplaçable	Entier long	5	
Form fenêtre barre outils	Entier long	35	
Form fenêtre feuille	Entier long	33	
Form fenêtre palette	Entier long	1984	
Form fenêtre pop up	Entier long	32	
Form fenêtre standard	Entier long	8	

Dictionnaires

Note de compatibilité : Ces constantes correspondent à des numéros de dictionnaires "Cordial", qui ne sont plus pris en charge dans 4D à compter de la version 14. Ces constantes sont conservées pour des raisons de compatibilité (un dictionnaire Hunspell de langue équivalente est utilisé en interne).

Constante	Type	Valeur	Comment

Documents système

Constante	Type	Valeur	Comment
Chemin absolu	Entier long	2	Le tableau *documents* contient des chemins d'accès absolus
Chemin POSIX	Entier long	4	Le tableau *documents* contient des chemins d'accès au format POSIX
Chemin récursif	Entier long	1	Le tableau *documents* contient les fichiers et tous les sous-dossiers du dossier spécifié
Document avec CR	Entier long	3	Les fins de ligne sont convertis au format OS X : CR (*carriage return*)
Document avec CRLF	Entier long	2	Les fins de ligne sont convertis au format Windows : CRLF (*carriage return + line feed*)
Document avec format natif	Entier long	1	(Défaut) Les fins de ligne sont convertis au format natif de la plate-forme d'exécution : CR (*carriage return*) sous OS X, CRLF (*carriage return + line feed*) sous Windows
Document avec LF	Entier long	4	Les fins de ligne sont convertis au format Unix : LF (*line feed*)
Document inchangé	Entier long	0	Aucun traitement
Est un document	Entier long	1	
Est un dossier	Entier long	0	
Fichiers multiples	Entier long	1	Autorise la sélection simultanée de plusieurs fichiers à l'aide des combinaisons **Maj+clic** (sélection contiguë) et **Ctrl+clic** (Windows) ou **Commande+clic** (Mac OS). Dans ce cas, le paramètre *sélectionnés*, s'il est passé, contient la liste de tous les fichiers sélectionnés. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas la sélection de plusieurs fichiers.
Ignorer invisibles	Entier long	8	Les documents invisibles ne sont pas listés
Lecture et écriture	Entier long	0	Mode par défaut
Lire chemin accès	Entier long	3	
Mode écriture	Entier long	1	
Mode lecture	Entier long	2	
Ouverture progiciel	Entier long	2	(Mac OS uniquement) Autorise l'ouverture des progiciels (packages) en tant que dossiers et donc la visualisation/sélection de leur contenu. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas l'ouverture des progiciels.
Saisie nom fichier	Entier long	32	Permet à l'utilisateur à saisir un nom de fichier dans une boîte de dialogue de sauvegarde. Aucun fichier n'est sauvegardé, il revient au développeur de créer un fichier en réponse à cette action (la variable système Document est mise à jour). Dans ce contexte, il est possible de passer un chemin de fichier dans le paramètre *répertoire*. Le nom du fichier sera suggéré dans la boîte de dialogue de sauvegarde et son répertoire parent sera utilisé comme chemin par défaut.
Sélection alias	Entier long	8	Autorise la sélection de raccourcis (Windows) ou d'alias (Mac OS) en tant que documents. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas la sélection de raccourcis ou d'alias en tant que tels. Si l'utilisateur sélectionne ce type de document, la commande retourne le chemin de l'élément cible. Lorsque vous passez la constante, la commande retourne le chemin de l'alias ou du raccourci lui-même.
Sélection progiciel	Entier long	4	(Mac OS uniquement) Autorise la sélection de progiciels (packages) en tant qu'entités. Par défaut, si cette constante n'est pas utilisée, la commande ne permet pas de sélectionner les progiciels en tant que tels.
Séparateur dossier	Chaîne	Windows="\" Mac OS=":"	Cette constante a une valeur différente en fonction du système d'exploitation depuis lequel elle est appelée. Elle permet de construire des chemins d'accès valides sans tenir compte de la plate-forme d'exécution. **Note :** Cette constante peut être utilisée uniquement dans les applications 4D exécutées en mode Unicode (elle n'est pas prise en charge en mode de compatibilité non-unicode).
Supprimer avec contenu	Entier long	1	Supprime le dossier ainsi que son éventuel contenu
Supprimer si vide	Entier long	0	Supprime le dossier uniquement s'il est vide
Utiliser fenêtre feuille	Entier long	16	(Mac OS uniquement) Affiche la boîte de dialogue de sélection sous forme de fenêtre feuille (cette option est ignorée sous Windows). Les fenêtres feuilles sont des fenêtres spécifiques de l'interface Mac OS X, bénéficiant notamment d'une animation graphique (pour plus d'informations, reportez-vous à la section **Types de fenêtres (compatibilité)**). Par défaut, si cette constante n'est pas utilisée, la commande affiche une boîte de dialogue standard.

Dossier Système

Constante	Type	Valeur	Comment
_o_Extensions Mac	Entier long	10	
_o_Ouverture extinction Mac	Entier long	7	
_o_Ouverture extinction Mac_Tou	Entier long	6	
_o_Tableaux de bord Mac	Entier long	11	
Applications ou Program Files	Entier long	16	
Bureau	Entier long	15	
Démarrage Win	Entier long	5	
Démarrage Win_Tous	Entier long	4	
Dossier documents	Entier long	17	Dossier "Documents" de l'utilisateur
Favoris Win	Entier long	14	
Menu Démarrer Win	Entier long	9	
Menu Démarrer Win_Tous	Entier long	8	
Polices	Entier long	1	
Préférences utilisateur	Entier long	3	
Préférences utilisateur_Tous	Entier long	2	
System Win	Entier long	12	
System32 Win	Entier long	13	
Système	Entier long	0	

Environnement 4D

Constante	Type	Valeur	Comment
_o_4D Interpreted Desktop	Entier long	2	
_o_Dossier Extras	Entier long	2	Dossier au contenu personnalisé téléchargé sur chaque poste client. **Note de compatibilité :** Depuis la version 11.2 de 4D v11 SQL, il est déconseillé d'utiliser le dossier **Extras** pour la communication personnalisée entre le serveur et les postes distants. Il est recommandé d'utiliser pour cela le dossier **Resources** (cf. description du dossier **Resources** courant ci-dessous). Le dossier **Extras** reste toutefois pris en charge par 4D Server afin de préserver la compatibilité des applications existantes. **Note :** Si le dossier **Extras** n'existe pas pour la base, l'exécution de la commande **Dossier 4D** avec la constante Dossier Extras provoque sa création.
_o_Version standard	Entier long	0	
4D Desktop	Entier long	3	
4D mode distant	Entier long	4	
4D mode local	Entier long	0	
4D Server	Entier long	5	
4D Volume Desktop	Entier long	1	
Application fusionnée	Entier long	2	La version est une application fusionnée avec 4D Volume Desktop
Dossier 4D actif	Entier long	0	
Dossier base	Entier long	4	
Dossier base 4D Client	Entier long	3	
Dossier base syntaxe Unix	Entier long	5	
Dossier données	Entier long	9	
Dossier Licenses	Entier long	1	
Dossier Logs	Entier long	7	
Dossier racine HTML	Entier long	8	
Dossier Resources courant	Entier long	6	
Fichier configuration sauvegarde	Entier long	1	Fichier Backup.xml, stocké dans le dossier Preferences/Backup à côté du fichier structure de la base.
Fichier dernière sauvegarde	Entier long	2	Dernier fichier de sauvegarde généré, nommé <NomBase>[NumBkp].4BK, stocké à un emplacement personnalisé.
Fichier propriétés structure	Entier long	3	settings.4DSettings pour tous les fichiers de données (si activé), stocké dans le dossier Preferences à côté du fichier de structure de la base
Fichier propriétés utilisateur pour données	Entier long	4	settings.4DSettings pour le fichier de données courant, stocké dans le dossier Preferences à côté du fichier de données.
Langue 4D interne	Entier long	3	Langue utilisée par 4D pour les tris et les comparaisons de textes (définie dans les Préférences de l'application).
Langue courante	Entier long	1	Langue courante de l'application : langue par défaut ou langue définie via la commande **FIXER LANGUE BASE**.
Langue par défaut	Entier long	0	Langue définie automatiquement par 4D au démarrage en fonction du dossier Resources et de l'environnement système (non modifiable).
Langue système	Entier long	2	Langue définie par l'utilisateur courant du système.
Propriétés structure	Entier long	0	Accès aux "propriétés structure" (valeur par défaut si le paramètre est omis). Dans ce mode, les valeurs de *sélecteur* utilisables sont identiques à celles du mode standard.
Propriétés utilisateur	Entier long	1	Accès aux "propriétés utilisateur". Dans ce mode, seules certaines clés sont utilisables dans le paramètre *sélecteur*.
Propriétés utilisateur pour données	Entier long	2	Accès aux "propriétés utilisateur pour données", c'est-à-dire les propriétés utilisateur stockées au même niveau que le fichier de données. Dans ce mode, seules certaines clés peuvent être utilisées avec le paramètre *sélecteur* (même sous-ensemble que les propriétés utilisateur).

Constante	Type	Valeur	Comment
Texte méthode	Entier long	1	
Texte méthode surligné	Entier long	2	
Version 64 bits	Entier long	1	
Version de démonstration	Entier long	0	

Euro monnaies

Constante	Type	Valeur	Comment
Drachme grecque	Chaîne	GRD	
Escudo portugais	Chaîne	PTE	
Euro	Chaîne	EUR	
Florin néerlandais	Chaîne	NLG	
Franc belge	Chaîne	BEF	
Franc français	Chaîne	FRF	
Franc luxembourgeois	Chaîne	LUF	
Lire italienne	Chaîne	ITL	
Livre irlandaise	Chaîne	IEP	
Mark allemand	Chaîne	DEM	
Mark finlandais	Chaîne	FIM	
Peseta espagnole	Chaîne	ESP	
Schilling autrichien	Chaîne	ATS	

Evénements (Modifiers)

Constante	Type	Valeur	Comment
Bit activation fenêtre	Entier long	0	
Bit bouton souris	Entier long	7	
Bit touche commande	Entier long	8	Touche Ctrl sous Windows, touche Commande sous OS X
Bit touche contrôle	Entier long	12	Touche Ctrl sous OS X, ou clic droit sous Windows et OS X
Bit touche contrôle droite	Entier long	15	
Bit touche majuscule	Entier long	9	Windows et OS X
Bit touche majuscule droite	Entier long	13	
Bit touche option	Entier long	11	Touche Alt (aussi appelée Option sous OS X)
Bit touche option droite	Entier long	14	
Bit touche verrouillage maj	Entier long	10	Windows et OS X
Masque activation fenêtre	Entier long	1	
Masque bouton souris	Entier long	128	
Masque touche commande	Entier long	256	Touche Ctrl sous Windows, touche Commande sous OS X
Masque touche contrôle	Entier long	4096	Touche Ctrl sous OS X, ou clic droit sous Windows et OS X
Masque touche contrôle droite	Entier long	32768	
Masque touche majuscule	Entier long	512	Windows et OS X
Masque touche majuscule droite	Entier long	8192	
Masque touche option	Entier long	2048	Touche Alt (aussi appelée Option sous OS X)
Masque touche option droite	Entier long	16384	
Masque touche verrouillage maj	Entier long	1024	Windows et OS X

Evénements (types)

Constante	Type	Valeur	Comment
Activation fenêtre	Entier long	8	
Bouton souris enfoncé	Entier long	1	
Bouton souris relâché	Entier long	2	
Evénement disque	Entier long	7	
Evénement nul	Entier long	0	
Evénement système	Entier long	15	
Mise à jour fenêtre	Entier long	6	
Répétition touche	Entier long	5	
Touche enfoncée	Entier long	3	
Touche relâchée	Entier long	4	

Evénements de la base

Constante	Type	Valeur	Comment
Sur après fermeture base hôte	Entier long	4	La **Méthode base Sur fermeture** de la base hôte vient de terminer son exécution
Sur après ouverture base hôte	Entier long	2	La **Méthode base Sur ouverture** de la base hôte vient de terminer son exécution
Sur avant fermeture base hôte	Entier long	3	La base hôte est en cours de fermeture. La **Méthode base Sur fermeture** de la base hôte n'a pas encore été appelée. La **Méthode base Sur fermeture** de la base hôte n'est pas appelée tant que la **Méthode base Sur événement base hôte** du composant est en exécution
Sur avant ouverture base hôte	Entier long	1	La base hôte vient juste d'être lancée. La **Méthode base Sur ouverture** de la base hôte n'a pas encore été appelée. La **Méthode base Sur ouverture** de la base hôte n'est pas appelée tant que la **Méthode base Sur événement base hôte** du composant est en exécution
Sur passage arrière plan	Entier long	1	L'application 4D passe à l'arrière plan
Sur passage premier plan	Entier long	2	L'application 4D passe au premier plan

Evénements formulaire

Constante	Type	Valeur	Comment
_o_Sur bouton barre outils Mac	Entier long	55	*** Constante obsolète ***
Sur action suppression	Entier long	58	(*Listes hiérarchiques et List box*) L'utilisateur a demandé à supprimer un élément
Sur activation	Entier long	11	La fenêtre du formulaire passe au premier plan
Sur affichage corps	Entier long	8	Un enregistrement va être affiché dans la liste ou une ligne va être affichée dans la list box
Sur appel extérieur	Entier long	10	Le formulaire a reçu un appel de la commande **APPELER PROCESS**
Sur appel zone du plug in	Entier long	19	Un plug-in demande que sa méthode objet soit exécutée
Sur après frappe clavier	Entier long	28	Un caractère vient d'être saisi dans l'objet qui a le focus. **Lire texte edite** retourne le contenu avec ce caractère.
Sur après modification	Entier long	45	Le contenu de l'objet saisissable qui a le focus vient d'être modifié
Sur après tri	Entier long	30	(*List box uniquement*) Un tri standard vient d'être effectué dans une colonne de list box
Sur avant frappe clavier	Entier long	17	Un caractère vient d'être saisi dans l'objet qui a le focus. **Lire texte edite** retourne le contenu sans ce caractère
Sur avant saisie	Entier long	41	(*List box uniquement*) Une cellule de list box est sur le point de passer en mode édition
Sur case de fermeture	Entier long	22	On a cliqué sur la case de fermeture de la fenêtre
Sur changement page	Entier long	56	On a changé de page courante dans le formulaire
Sur chargement	Entier long	1	Le formulaire s'affiche ou s'imprime
Sur chargement ligne	Entier long	40	En mode saisie en liste, un enregistrement est chargé en modification (l'utilisateur a cliqué sur la ligne de l'enregistrement et un champ passe en édition)
Sur chargement ressource URL	Entier long	48	(*Zones Web uniquement*) Une nouvelle ressource est chargée dans la zone Web
Sur clic	Entier long	4	Un clic est survenu sur un objet
Sur clic alternatif	Entier long	38	• *Boutons 3D* : La zone "flèche" d'un bouton 3D reçoit un clic • *List box* : Dans une colonne tableau d'objets, un bouton d'ellipse (attribut "alternateButton") reçoit un clic **Note**: Les boutons d'ellipses sont disponibles à partir de la v15 uniquement.
Sur clic entête	Entier long	42	(*List box uniquement*) Un clic est survenu dans l'en-tête d'une colonne de list box
Sur clic long	Entier long	39	(*Boutons 3D uniquement*) Un bouton 3D reçoit un clic et le bouton de la souris reste enfoncé pendant un certain laps de temps
Sur clic pied	Entier long	57	(*List box uniquement*) Un clic est survenu dans le pied d'une list box ou d'une colonne de list box
Sur contracter	Entier long	44	(*Listes hiérarchiques et list box hiérarchiques*) Un élément de liste hiérarchique ou de list box hiérarchique a été contracté via un clic ou une touche du clavier
Sur début chargement URL	Entier long	47	(*Zones Web uniquement*) Un nouvel URL est chargé dans la zone Web
Sur début glisser	Entier long	46	Un objet est en cours de glisser
Sur début survol	Entier long	35	Le curseur de la souris entre dans la zone graphique d'un objet
Sur défilement	Entier long	59	*Variables ou champs image et List Box* : L'utilisateur fait défiler le contenu du champ image, de la variable image ou de la list box à l'aide de la souris ou d'une touche du clavier.
Sur déplacement colonne	Entier long	32	(*List box uniquement*) Une colonne de list box est déplacée par l'utilisateur via le glisser-déposer
Sur déplacement ligne	Entier long	34	(*List box uniquement*) Une ligne de list box est déplacée par l'utilisateur via le glisser-déposer
Sur déployer	Entier long	43	(*Listes hiérarchiques et List box hiérarchiques*) Un élément de liste hiérarchique ou de list box hiérarchique a été déployé via un clic ou une touche du clavier
Sur déposer	Entier long	16	Des données sont déposées sur un objet
Sur désactivation	Entier long	12	La fenêtre du formulaire passe en arrière-plan
Sur données modifiées	Entier long	20	Les données d'un objet ont été modifiées
Sur double clic	Entier long	13	Un double-clic est survenu sur un objet
Sur entête	Entier long	5	L'en-tête du formulaire va être imprimé ou affiché

Constante	Type	Valeur	Comment
Sur erreur chargement URL	Entier long	50	(*Zones Web uniquement*) Une erreur s'est produite durant le chargement de l'URL
Sur fermeture corps	Entier long	26	Le formulaire détaillé se referme et on retourne au formulaire sortie
Sur filtrage URL	Entier long	51	(*Zones Web uniquement*) Un URL a été bloqué par la zone Web
Sur fin chargement URL	Entier long	49	(*Zones Web uniquement*) Toutes les ressources de l'URL ont été chargées
Sur fin survol	Entier long	36	Le curseur de la souris sort de la zone graphique d'un objet
Sur gain focus	Entier long	15	Un objet de formulaire prend le focus
Sur glisser	Entier long	21	Des données sont glissées sur un objet
Sur impression corps	Entier long	23	Le corps du formulaire va être imprimé
Sur impression pied de page	Entier long	7	Le pied de page du formulaire va être imprimé
Sur impression sous total	Entier long	6	Une rupture du formulaire va être imprimée
Sur libération	Entier long	24	Le formulaire se referme et est déchargé
Sur menu sélectionné	Entier long	18	Une commande de menu a été sélectionnée
Sur minuteur	Entier long	27	Le nombre de ticks défini par **FIXER MINUTEUR** est atteint
Sur modif variable liée	Entier long	54	La variable liée à un sous-formulaire est modifiée.
Sur nouvelle sélection	Entier long	31	• *List box* : la sélection courante de lignes ou de colonnes est modifiée • *Enregistrements en liste* : l'enregistrement courant ou la sélection courante de lignes est modifié(e) dans un formulaire en liste ou un sous-formulaire • *Liste hiérarchique* : la sélection dans la liste est modifiée à la suite d'un clic ou de la frappe d'une touche au clavier • *Variable ou champ saisissable* : la sélection de texte ou la position du curseur dans la zone est modifiée à la suite d'un clic ou de la frappe d'une touche au clavier
Sur ouverture corps	Entier long	25	Le formulaire détaillé associé au formulaire sortie ou à la listbox est sur le point d'être ouvert
Sur ouverture lien externe	Entier long	52	(*Zones Web uniquement*) Un URL externe a été ouvert dans le navigateur
Sur perte focus	Entier long	14	Un objet de formulaire perd le focus
Sur redimensionnement	Entier long	29	La fenêtre du formulaire est redimensionnée
Sur redimensionnement colonne	Entier long	33	(*List box uniquement*) La largeur d'une colonne de list box est modifiée par l'utilisateur via la souris
Sur refus ouverture fenêtre	Entier long	53	(*Zones Web uniquement*) Une fenêtre pop up a été bloquée
Sur relâchement bouton	Entier long	2	(*Images uniquement*) L'utilisateur vient de relâcher le bouton de la souris dans un objet Image
Sur survol	Entier long	37	Le curseur de la souris a bougé d'au moins un pixel OU une touche de modification (Ctrl, Alt, Verr Maj.) a été enfoncée. Si l'événement est coché pour un objet uniquement, il n'est généré que lorsque le curseur se trouve dans la zone graphique de l'objet
Sur validation	Entier long	3	La saisie des données dans l'enregistrement est validée

Evénements trigger

Constante	Type	Valeur	Comment
_o_Sur chargement enreg	Entier long	4	
Sur sauvegarde enregistrement	Entier long	2	
Sur sauvegarde nouvel enreg	Entier long	1	
Sur suppression enregistrement	Entier long	3	

Expressions

Constante	Type	Valeur	Comment
MAXENT	Entier long	32767	
MAXLONG	Entier long	2147483647	
MAXLONGTEXTEAVANTV11	Entier long	32000	

Fenêtre

Constante	Type	Valeur	Comment
Fenêtre externe	Entier long	5	
Fenêtre flottante	Entier long	14	
Fenêtre modale	Entier long	9	
Fenêtre normale	Entier long	8	
XY Ecran	Entier long	3	L'origine est le coin supérieur de l'écran principal (comme pour la commande **COORDONNEES ECRAN**)
XY Fenêtre courante	Entier long	2	L'origine est le coin supérieur gauche de la fenêtre courante
XY Fenêtre principale	Entier long	4	Windows : L'origine est le coin supérieur gauche de la fenêtre principale ; OS X : identique à XY Ecran
XY Formulaire courant	Entier long	1	L'origine est le coin supérieur gauche du formulaire courant

FIXER COULEUR RVB

Constante	Type	Valeur	Comment
Coul arrière plan	Entier long	-2	
Coul claire	Entier long	-4	
Coul de fond texte sélect	Entier long	-7	
Coul fond élément sélect désact	Entier long	-11	
Coul fond ligne menu sélect	Entier long	-9	
Coul fond transparent	Entier long	-16	Cette constante peut être utilisée uniquement avec les paramètres *couleurArrièrePlan* et *couleurArrièrePlanAlt*.
Coul premier plan	Entier long	-1	
Coul sombre	Entier long	-3	
Coul texte ligne menu sélect	Entier long	-10	
Coul texte sélect	Entier long	-8	

Fonctions mathématiques

Constante	Type	Valeur	Comment
Degré	Réel	0.0174532925199432958	
Nombre e	Réel	2.71828182845904524	
Pi	Réel	3.141592653589793239	
Radian	Réel	57.29577951308232088	

Formatages système

Constante	Type	Valeur	Comment
Libellé AM heure système	Entier long	18	Libellé additionnel des heures d'avant midi dans les formats sur 12 heures (ex : "Matin")
Libellé PM heure système	Entier long	19	Libellé additionnel des heures après midi dans les formats sur 12 heures (ex : "Après-Midi")
Motif date abrégé	Entier long	7	Format d'affichage de date abrégé sous la forme "dddd MMMM yyyy"
Motif date court	Entier long	6	Format d'affichage de date court sous la forme "dddd MMMM yyyy"
Motif date long	Entier long	8	Format d'affichage de date long sous la forme "dddd MMMM yyyy"
Motif heure abrégé	Entier long	4	Format d'affichage d'heure abrégé sous la forme "HH:mm:ss"
Motif heure court	Entier long	3	Format d'affichage d'heure court sous la forme "HH:mm:ss"
Motif heure long	Entier long	5	Format d'affichage d'heure long sous la forme "HH:mm:ss"
Position année date courte	Entier long	17	Position de l'année dans le format de date court : "1" = à gauche, "2" = au milieu, "3" = à droite
Position jour date courte	Entier long	15	Position du jour dans le format de date court : "1" = à gauche, "2" = au milieu, "3" = à droite
Position mois date courte	Entier long	16	Position du mois dans le format de date court : "1" = à gauche, "2" = au milieu, "3" = à droite
Séparateur date	Entier long	13	Séparateur utilisé dans les formats de dates (ex : "/")
Séparateur de milliers	Entier long	1	Séparateur de milliers (ex : " ")
Séparateur décimal	Entier long	0	Séparateur décimal (ex : ",")
Séparateur heure	Entier long	14	Séparateur utilisé dans les formats d'heures (ex : ":")
Symbole monétaire	Entier long	2	Symbole monétaire (ex : "$")

Formats d'affichage des dates

Constante	Type	Valeur	Comment
Date RFC 1123	Entier long	10	Fri, 10 Sep 2010 13:07:20 GMT
Interne date abrégé	Entier long	6	6 déc 1996
Interne date court	Entier long	7	06/12/2006
Interne date court spécial	Entier long	4	06/12/06 (mais 06/12/1896 ou 06/12/2096)
Interne date long	Entier long	5	6 décembre 2006
ISO Date	Entier long	8	2006-06-12T00:00:00 (format obsolète)
ISO Date GMT	Entier long	9	2010-09-13T16:11:53Z
Système date abrégé	Entier long	2	mer. 25 déc. 2006
Système date court	Entier long	1	06/12/2006
Système date long	Entier long	3	mercredi 6 décembre 2006
Vide si date nulle	Entier long	100	"" au lieu de 0 en cas de valeur nulle. Cette constante doit être additionnée au format d'affichage.

Formats d'affichage des heures

Constante	Type	Valeur	Comment
h mn	Entier long	2	01:02
h mn Matin Après Midi	Entier long	5	1:02 du matin
h mn s	Entier long	1	01:02:03
Heures minutes	Entier long	4	1 heure 2 minutes
Heures minutes secondes	Entier long	3	1 heure 2 minutes 3 secondes
ISO heure	Entier long	8	0000-00-00T01:02:03
Minutes secondes	Entier long	7	62 minutes 3 secondes
mn s	Entier long	6	62:03
Système heure court	Entier long	9	01:02:03
Système heure long	Entier long	11	1:02:03 AM HNEC (Mac uniquement)
Système heure long abrégé	Entier long	10	1•02•03 AM (Mac uniquement)
Vide si heure nulle	Entier long	100	"" au lieu de 0

Formats d'affichage des images

Constante	Type	Valeur	Comment
Mosaïque	Entier long	7	
Non tronquée	Entier long	2	
Proportionnelle	Entier long	5	
Proportionnelle centrée	Entier long	6	
Sur fond	Entier long	3	
Tronquée centrée	Entier long	1	
Tronquée non centrée	Entier long	4	

Interfaces de plate-forme

Constante	Type	Valeur	Comment

Journal d'événements

Constante	Type	Valeur	Comment
Message d'erreur	Entier long	2	
Message d'avertissement	Entier long	1	
Message d'information	Entier long	0	
Vers historique commandes 4D	Entier long	3	Indique à 4D d'inscrire le *message* dans le fichier d'historique des commandes de 4D, si ce fichier a été activé. Ce fichier d'historique peut être activé à l'aide de la commande **FIXER PARAMETRE BASE** (sélecteur 34). **Note :** Les fichiers d'historique de 4D sont regroupés dans le dossier **Logs**, créé à côté du fichier de structure de la base (cf. commande **Dossier 4D**).
Vers historique diagnostic	Entier long	5	Indique à 4D d'inscrire le *message* dans le fichier de diagnostic de 4D, si ce fichier a été activé. Le fichier de diagnostic peut être activé à l'aide de la commande **FIXER PARAMETRE BASE** (sélecteur 79).
Vers historique requêtes 4D	Entier long	2	Indique à 4D d'inscrire le *message* dans le fichier d'historique des requêtes de 4D, si ce fichier a été activé
Vers message débogage	Entier long	1	Indique à 4D d'envoyer le *message* dans l'environnement de débogage du système. Le résultat dépend de la plate-forme : • sous Mac OS : la commande envoie le message à la Console • sous Windows : la commande envoie le message en tant que message de débogage. Pour pouvoir lire ce message, vous devez disposer de Microsoft Visual Studio ou de l'utilitaire DebugView pour Windows (http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx)
Vers observateur Windows	Entier long	0	Indique à 4D d'envoyer le *message* vers l'"Observateur d'événements" de Windows. Ce journal reçoit et stocke des messages en provenance des applications en cours d'exécution. Dans ce cas, vous pouvez définir le niveau d'importance du *message* via le paramètre *importance* (cf. ci-dessous). **Notes :** • Pour que cette fonctionnalité soit disponible, le service Observateur d'événements de Windows doit être démarré. • Sous Mac OS, la commande ne fait rien avec ce type de sortie.

Jours et mois

Constante	Type	Valeur	Comment
Août	Entier long	8	
Avril	Entier long	4	
Décembre	Entier long	12	
Dimanche	Entier long	1	
Février	Entier long	2	
Janvier	Entier long	1	
Jeudi	Entier long	5	
Juillet	Entier long	7	
Juin	Entier long	6	
Lundi	Entier long	2	
Mai	Entier long	5	
Mardi	Entier long	3	
Mars	Entier long	3	
Mercredi	Entier long	4	
Novembre	Entier long	11	
Octobre	Entier long	10	
Samedi	Entier long	7	
Septembre	Entier long	9	
Vendredi	Entier long	6	

LDAP

Constante	Type	Valeur	Comment
LDAP mot de passe en clair	Entier long	1	Envoi du mot de passe sans encryptage (connexion TLS recommandée)
LDAP mot de passe en digest MD5	Entier long	0	(Défaut) Envoi du mot de passe encrypté en MD5
LDAP racine et suivant	Chaîne	one	Chercher dans l'élément racine défini par *dnRootEntry* et dans les branches directement suivantes sur un niveau
LDAP racine uniquement	Chaîne	base	Chercher uniquement dans l'élément racine défini par *dnRootEntry* (défaut si omis)
LDAP tous niveaux	Chaîne	sub	Chercher dans l'élément racine défini par *dnRootEntry* et dans toutes les branches suivantes

Licence disponible

Constante	Type	Valeur	Comment
Licence 4D for OCI	Entier long	808465208	
Licence 4D Mobile	Entier long	808464439	
Licence 4D ODBC Pro	Entier long	808464946	
Licence 4D SOAP	Entier long	808465464	
Licence 4D SOAP locale	Entier long	808531000	
Licence 4D SOAP une connexion	Entier long	825242680	
Licence 4D View	Entier long	808465207	
Licence 4D Web	Entier long	808464945	
Licence 4D Web locale	Entier long	808530481	
Licence 4D Web une connexion	Entier long	825242161	
Licence 4D Write	Entier long	808464697	
Licence Serveur SQL 4D	Entier long	808464949	
Licence Serveur SQL 4D 1 conn	Entier long	825242165	
Licence Serveur SQL 4D locale	Entier long	808530485	
Licence SOAP 4D Client	Entier long	808465465	
Licence Test 4D Mobile	Entier long	808465719	
Licence Web 4D Client	Entier long	808465209	

Liens

Constante	Type	Valeur	Comment
Automatique	Entier long	3	
Configuration structure	Entier long	1	
Manuel	Entier long	2	
Ne pas changer	Entier long	0	
Pas de lien	Entier long	0	

List box

Constante	Type	Valeur	Comment
lk affichage barre déf hor	Entier long	2	0=masquée, 1=affichée
lk affichage barre déf ver	Entier long	4	0=masquée, 1=affichée
lk affichage entête	Entier long	0	Propriété **Afficher en-têtes** S'applique à : List box Valeurs possibles : • lk non (0) : masqué • lk oui (1) : affiché
lk affichage pied	Entier long	8	Propriété **Afficher pieds** S'applique à : List box Valeurs possibles : • lk non (0) : masqué • lk oui (1) : affiché
lk ajouter à sélection	Entier long	1	La ligne sélectionnée est ajoutée à la sélection existante. Si la ligne désignée appartient déjà à la sélection existante, la commande ne fait rien.
lk couleur de fond	Entier long	1	
lk couleur de police	Entier long	0	
lk hauteur barre déf hor	Entier long	3	Hauteur en pixels
lk hauteur entête	Entier long	1	Hauteur en pixels
lk hauteur imprimée	Entier long	3	Retourne dans *info* la hauteur en pixels de l'objet effectivement imprimé (inclut les en-têtes, filets, etc.). Souvenez-vous que si le nombre de lignes à imprimer est inférieur à la "capacité" de la list box, sa hauteur est automatiquement réduite.
lk hauteur pied	Entier long	9	Hauteur en pixels
lk hérité	Entier long	-255	
lk impression terminée	Entier long	2	Retourne dans *info* un booléen indiquant si la dernière ligne (visible) de la list box a été effectivement imprimée. Vrai = la ligne a été imprimée, Faux sinon.
lk largeur barre déf ver	Entier long	5	Largeur en pixels
lk ligne désactivée	Entier long	2	La ligne correspondante est désactivée. Les textes et les contrôles tels que les cases à cocher sont grisés ou estompés. Les zones de texte ne sont plus saisissables. Par défaut : activée
lk ligne masquée	Entier long	1	La ligne correspondante est masquée. Masquer des lignes affecte uniquement l'affichage de la list box. Les lignes masquées sont toujours présentes dans les tableaux et peuvent être manipulées par programmation. Les commandes du langage, notamment **LISTBOX Lire nombre lignes** ou **LISTBOX LIRE POSITION CELLULE**, ne tiennent pas compte de l'état masqué/affiché des lignes. Par exemple, dans une list box contenant 10 lignes et dont les 9 premières sont masquées, **LISTBOX Lire nombre lignes** retournera 10. Du point de vue de l'utilisateur, la présence de lignes masquées dans une list box n'est pas décelable visuellement. Seules les lignes visibles sont sélectionnables (par exemple via la commande Tout sélectionner). Par défaut : visible
lk ligne non sélectionnable	Entier long	4	La ligne correspondante n'est pas sélectionnable (le surlignage n'est plus possible). Les zones de texte ne sont plus saisissables à moins que l'option "Saisie sur clic unique" soit active. Les contrôles tels que les cases à cocher et les pop ups restent toutefois fonctionnels. Ce paramétrage est ignoré si le mode de sélection de la list box est "Aucun". Par défaut : sélectionnable
lk ligne rupture	Entier long	2	La commande agit sur le sous-niveau auquel appartient la "cellule" désignée par les paramètres *ligne* et *colonne*. A noter que ces paramètres représentent les numéros de ligne et de colonne dans la listbox en mode standard et non dans sa représentation hiérarchique. Si les paramètres *ligne* et *colonne* sont omis, la commande ne fait rien.
lk lignes	Entier long	1	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police.
lk niveau	Entier long	3	La commande agit sur toutes les lignes de rupture correspondant à la colonne *niveau*. Ce paramètre désigne un numéro de colonne dans la listbox en mode standard et non dans sa représentation hiérarchique. Si le paramètre *niveau* est omis, la commande ne fait rien.
lk nombre lignes imprimées	Entier long	1	Retourne dans *info* le nombre de lignes effectivement imprimées lors du dernier appel à la commande **Imprimer objet**. Ce nombre inclut les éventuelles lignes de ruptures ajoutées dans le cadre d'une list box hiérarchique. Par exemple, *info* vaut 10 si la list box contient 20 lignes et que les lignes impaires ont été masquées.
lk num dernière ligne impr	Entier long	0	Retourne dans *info* le numéro de la dernière ligne imprimée. Permet de connaître le numéro de la prochaine ligne à imprimer. Le numéro retourné peut être supérieur au nombre de lignes effectivement imprimées si la list box contient des lignes invisibles ou si la commande **OBJET FIXER DEFILEMENT** a été appelée. Par exemple, si les lignes 1, 18 et 20 ont été imprimées, *info* vaut 20.
lk pixels	Entier long	0	La hauteur est un nombre de pixels (défaut)
lk position barre déf hor	Entier long	6	Position du curseur en pixels
lk position barre déf ver	Entier long	7	Position du curseur en pixels
lk remplacer sélection	Entier long	0	La ligne sélectionnée devient la nouvelle sélection et remplace la sélection existante. La commande produit le même effet qu'un clic de l'utilisateur sur une ligne de la list box (l'événement Sur clic n'est toutefois pas généré). Cette action est effectuée par défaut (lorsque le paramètre *action* n'est pas passé).

Constante	Type	Valeur	Comment
lk sélection	Entier long	1	La commande agit sur les sous-niveaux sélectionnés.
lk supprimer de sélection	Entier long	2	La ligne sélectionnée est supprimée de la sélection existante. Si la ligne désignée n'appartient pas à la sélection existante, la commande ne fait rien.
lk tableau contrôle	Entier long	3	
lk tableau couleur de fond	Entier long	1	
lk tableau couleur de police	Entier long	0	
lk tableau hauteurs lignes	Entier long	4	(Licence 4D View Pro requise)
lk tableau style	Entier long	2	
lk tout	Entier long	0	La commande agit sur tous les sous-niveaux (valeur par défaut, utilisée si le paramètre est omis).

List box pied calcul

Constante	Type	Valeur	Comment
lk pied écart type	Entier long	7	Utilisable avec les colonnes de type numérique, heure (listbox de type tableau uniquement) Type par défaut du résultat du calcul : Réel
lk pied max	Entier long	3	Utilisable avec les colonnes de type numérique, date, heure, booléen Type par défaut du résultat du calcul : type du tableau ou champ de la colonne
lk pied min	Entier long	2	Utilisable avec les colonnes de type numérique, date, heure, booléen Type par défaut du résultat du calcul : type du tableau ou champ de la colonne
lk pied moyenne	Entier long	6	Utilisable avec les colonnes de type numérique, heure Type par défaut du résultat du calcul : Réel
lk pied nombre	Entier long	5	Utilisable avec les colonnes de type numérique, texte, date, heure, booléen, image Type par défaut du résultat du calcul : Entier long
lk pied personnalisé	Entier long	1	Aucun calcul n'est effectué par 4D. La variable du pied doit être calculée par programmation. Type par défaut du résultat du calcul : type de la variable
lk pied somme	Entier long	4	Utilisable avec les colonnes de type numérique, heure, booléen Type par défaut du résultat du calcul : type du tableau ou champ de la colonne
lk pied somme des carrés	Entier long	9	Utilisable avec les colonnes de type numérique, heure (listbox de type tableau uniquement) Type par défaut du résultat du calcul : Réel
lk pied variance	Entier long	8	Utilisable avec les colonnes de type numérique, heure (listbox de type tableau uniquement) Type par défaut du résultat du calcul : Réel

Listes hiérarchiques

Constante	Type	Valeur	Comment
_o_Réf icône Macintosh	Entier long	860	
_o_Réf icône Windows	Entier long	138	
_o_Utiliser ressource PICT	Entier long	65536	
Texte supplémentaire	Chaîne	4D_additional_text	
Utiliser réf image	Entier long	131072	

Maintenance fichier de données

Constante	Type	Valeur	Comment
Compacter table adresses	Entier long	131072	Forcer la réécriture de la table d'adresses des enregistrements (ralentit le compactage). A noter que dans ce cas, les numéros des enregistrements sont réécrits. Si vous passez uniquement cette option, 4D active automatiquement l'option 'Mettre à jour enregistrements'.
Créer un process	Entier long	32768	Lorsque cette option est passée, le compactage sera asynchrone et vous devrez gérer les résultats à l'aide de la méthode de rétro-appel (voir ci-dessous). 4D n'affichera pas de barre de progression (il est possible de le faire via la méthode de rétro-appel). La variable système OK prendra la valeur 1 si le process a été correctement lancé et 0 dans les autres cas. Lorsque cette option n'est pas passée, la variable OK prendra la valeur 1 si le compactage s'est correctement déroulé et sinon 0.
Déplacer dans Replaced files	Entier long	4	
Dialogue nouveau fichier	Entier long	1	
Mettre à jour enregistrements	Entier long	65536	Forcer la réécriture de tous les enregistrements suivant la définition courante des champs dans la structure
Ne pas compacter les index	Entier long	2	
Ne pas créer d'historique	Entier long	16384	En principe, la commande crée un fichier d'historique au format xml (reportez-vous à la fin de la description de la commande). Vous pouvez annuler ce fonctionnement en passant cette option.
Nom historique avec date heure	Entier long	262144	Lorsque cette option est passée, le nom du fichier d'historique généré contiendra la date et l'heure de sa création, par conséquent il ne remplacera aucun fichier d'historique éventuellement déjà généré. Par défaut, si cette option n'est pas passée, le nom du fichier n'est pas horodaté, et chaque nouveau fichier généré remplace le précédent.
Nouveau fichier	Entier long	0	
Renuméroter les enregistrements	Entier long	1	
Tout vérifier	Entier long	16	
Utiliser définition structure	Entier long	0	4D utilisera les paramètres définis dans la structure pour le stockage du champ (cf. manuel *Mode Développement*). Si vous passez d'un stockage externe à un stockage interne, le fichier externe n'est pas supprimé.
Utiliser dossier par défaut	Entier long	1	Les données du champ passé en paramètre seront stockées dans le dossier par défaut, nommé *nomBase*.ExternalData et placé à côté du fichier de données. Dans ce mode, les données externes sont gérées par 4D comme si elles étaient à l'intérieur du fichier de données.
Utiliser fichier sélectionné	Entier long	2	
Vérifier enregistrements	Entier long	4	
Vérifier index	Entier long	8	Cette option contrôle la cohérence physique des index, sans lien avec les données. Elle signale des clés invalides mais ne permet pas de détecter les clés dupliquées (deux index pointant vers le même enregistrement). Ce type d'erreur ne peut être détecté qu'avec l'option Tout vérifier.

Moteur de la base

Constante	Type	Valeur	Comment
Aucun enregistrement courant	Entier long	-1	
Est un nouvel enregistrement	Entier long	-3	

Noms des métadonnées images

Constante	Type	Valeur	Comment
EXIF aperture value	Chaîne	EXIF/ApertureValue	Valeurs possibles : Réel (valeur APEX)
EXIF brightness value	Chaîne	EXIF/BrightnessValue	Valeurs possibles : Réel (valeur APEX)
EXIF color space	Chaîne	EXIF/ColorSpace	Valeurs possibles : EXIF Adobe RGB, EXIF s RGB, EXIF Uncalibrated (lecture seulement)
EXIF components configuration	Chaîne	EXIF/ComponentsConfiguration	Valeurs possibles : EXIF B, EXIF Cb, EXIF Cr, EXIF G, EXIF R, EXIF Unused, EXIF Y
EXIF compressed bits per pixel	Chaîne	EXIF/CompressedBitsPerPixel	Valeurs possibles : Réel
EXIF contrast	Chaîne	EXIF/Contrast	Valeurs possibles : EXIF High, EXIF Low, EXIF Normal
EXIF custom rendered	Chaîne	EXIF/CustomRendered	Valeurs possibles : EXIF Normal, EXIF Custom
EXIF date time digitized	Chaîne	EXIF/DateTimeDigitized	Valeurs possibles : Date ou Texte (Datetime XML)
EXIF date time original	Chaîne	EXIF/DateTimeOriginal	Valeurs possibles : Date ou Texte (Datetime XML)
EXIF digital zoom ratio	Chaîne	EXIF/DigitalZoomRatio	Valeurs possibles : Réel
EXIF EXIF version	Chaîne	EXIF/ExifVersion	Valeurs possibles : Tableau Entier (4 valeurs)
EXIF exposure bias value	Chaîne	EXIF/ExposureBiasValue	Valeurs possibles : Réel
EXIF exposure index	Chaîne	EXIF/ExposureIndex	Valeurs possibles : Réel
EXIF exposure mode	Chaîne	EXIF/ExposureModus	Valeurs possibles : EXIF Auto, EXIF Auto Bracket, EXIF Manual
EXIF exposure program	Chaîne	EXIF/ExposureProgram	Valeurs possibles : EXIF Manual, EXIF Action, EXIF Aperture Priority AE, EXIF Creative, EXIF Landscape, EXIF Exposure Portrait, EXIF Program AE, EXIF Shutter Speed Priority AE
EXIF exposure time	Chaîne	EXIF/ExposureTime	Valeurs possibles : Réel
EXIF F number	Chaîne	EXIF/FNumber	Valeurs possibles : Réel
EXIF file source	Chaîne	EXIF/FileSource	Valeurs possibles : EXIF Digital Camera, EXIF Film Scanner, EXIF Reflection Print Scanner
EXIF flash	Chaîne	EXIF/Flash	Valeurs possibles : EXIF Auto Mode, EXIF Compulsory Flash Firing, EXIF Compulsory Flash Suppression, EXIF Unknown, EXIF Detected, EXIF No Detection Function, EXIF Not Detected, EXIF Reserved
EXIF flash energy	Chaîne	EXIF/FlashEnergy	Valeurs possibles : Réel
EXIF flash fired	Chaîne	EXIF/Flash/Fired	Valeurs possibles : Booléen
EXIF flash function present	Chaîne	EXIF/Flash/FunctionPresent	Valeurs possibles : Booléen
EXIF flash mode	Chaîne	EXIF/Flash/Mode	Valeurs possibles : EXIF Auto Mode, EXIF Compulsory Flash Firing, EXIF Compulsory Flash Suppression, EXIF Unknown
EXIF flash pix version	Chaîne	EXIF/FlashPixVersion	Valeurs possibles : Tableau Entier (4 valeurs)
EXIF flash red eye reduction	Chaîne	EXIF/Flash/RedEyeReduction	Valeurs possibles : Booléen
EXIF flash return light	Chaîne	EXIF/Flash/ReturnLight	Valeurs possibles : EXIF Detected, EXIF No Detection Function, EXIF Not Detected, EXIF Reserved
EXIF focal len in 35 mm film	Chaîne	EXIF/FocalLenIn35mmFilm	Valeurs possibles : Entier long
EXIF focal length	Chaîne	EXIF/FocalLength	Valeurs possibles : Réel
EXIF focal plane resolution unit	Chaîne	EXIF/FocalPlaneResolutionUnit	Valeurs possibles : Entier long

Constante	Type	Valeur	Commentaire
EXIF focal plane X resolution	Chaîne	EXIF/FocalPlaneXResolution	Valeurs possibles : Réel
EXIF focal plane Y resolution	Chaîne	EXIF/FocalPlaneYResolution	Valeurs possibles : Réel
EXIF gain control	Chaîne	EXIF/GainControl	Valeurs possibles : EXIF High Gain Down, EXIF High Gain Up, EXIF Low Gain Down, EXIF Low Gain Up, EXIF None
EXIF gamma	Chaîne	EXIF/Gamma	Valeurs possibles : Réel
EXIF image unique ID	Chaîne	EXIF/ImageUniqueID	Valeurs possibles : Texte
EXIF ISO speed ratings	Chaîne	EXIF/ISOSpeedRatings	Valeurs possibles : Entier long ou Tableau Entier long
EXIF light source	Chaîne	EXIF/LightSource	Valeurs possibles : EXIF Unknown, EXIF Cloudy, EXIF Cool White Fluorescent, EXIF D50, EXIF D55, EXIF D65, EXIF D75, EXIF Daylight, EXIF Daylight Fluorescent, EXIF Day White Fluorescent, EXIF Fine Weather, EXIF Flashlight, EXIF Light Fluorescent, EXIF ISOStudio Tungsten, EXIF Other, EXIF Shade, EXIF Standard Light A, EXIF Standard Light B, EXIF Standard Light C, EXIF Tungsten, EXIF White Fluorescent
EXIF maker note	Chaîne	EXIF/MakerNote	Valeurs possibles : Texte
EXIF max aperture value	Chaîne	EXIF/MaxApertureValue	Valeurs possibles : Réel
EXIF metering mode	Chaîne	EXIF/MeteringMode	Valeurs possibles : EXIF Other, EXIF Average, EXIF Center Weighted Average, EXIF Multi Segment, EXIF Multi Spot, EXIF Partial, EXIF Spot
EXIF pixel X dimension	Chaîne	EXIF/PixelXDimension	Valeurs possibles : Entier long (lecture seulement)
EXIF pixel Y dimension	Chaîne	EXIF/PixelYDimension	Valeurs possibles : Entier long (lecture seulement)
EXIF related sound file	Chaîne	EXIF/RelatedSoundFile	Valeurs possibles : Texte
EXIF saturation	Chaîne	EXIF/Saturation	Valeurs possibles : EXIF High, EXIF Low, EXIF Normal
EXIF scene capture type	Chaîne	EXIF/SceneCaptureType	Valeurs possibles : EXIF Scene Landscape, EXIF Night, EXIF Scene Portrait, EXIF Standard
EXIF scene type	Chaîne	EXIF/SceneType	Valeurs possibles : Entier long
EXIF sensing method	Chaîne	EXIF/SensingMethod	Valeurs possibles : EXIF Color Sequential Area, EXIF Color Sequential Linear, EXIF Not Defined, EXIF One Chip Color Area, EXIF Three Chip Color Area, EXIF Trilinear, EXIF Two Chip Color Area
EXIF sharpness	Chaîne	EXIF/Sharpness	Valeurs possibles : EXIF High, EXIF Low, EXIF Normal
EXIF shutter speed value	Chaîne	EXIF/ShutterSpeedValue	Valeurs possibles : Réel
EXIF spectral sensitivity	Chaîne	EXIF/SpectralSensitivity	Valeurs possibles : Texte
EXIF subject area	Chaîne	EXIF/SubjectArea	Valeurs possibles : Tableau Entier long (2, 3 ou 4 valeurs)
EXIF subject dist range	Chaîne	EXIF/SubjectDistRange	Valeurs possibles : EXIF Unknown, EXIF Close, EXIF Distant, EXIF Macro
EXIF subject distance	Chaîne	EXIF/SubjectDistance	Valeurs possibles : Réel
EXIF subject location	Chaîne	EXIF/SubjectLocation	Valeurs possibles : Tableau Entier long (2 valeurs)
EXIF user comment	Chaîne	EXIF/UserComment	Valeurs possibles : Texte
EXIF white balance	Chaîne	EXIF/WhiteBalance	Valeurs possibles : EXIF Auto, EXIF Manual
GPS altitude	Chaîne	GPS/Altitude	Valeurs possibles : GPS Above Sea Level, GPS Below Sea Level
GPS altitude ref	Chaîne	GPS/AltitudeRef	Valeurs possibles : GPS Above Sea Level, GPS Below Sea Level
GPS area information	Chaîne	GPS/AreaInformation	Valeurs possibles : Texte
GPS date time	Chaîne	GPS/DateTime	Valeurs possibles : Date ou Texte (Datetime XML)
GPS dest bearing	Chaîne	GPS/DestBearing	Valeurs possibles : Texte (1 caractère)
GPS dest bearing ref	Chaîne	GPS/DestBearingRef	Valeurs possibles : Texte (1 caractère)
GPS dest distance	Chaîne	GPS/DestDistance	Valeurs possibles : Texte (1 caractère)
GPS dest distance ref	Chaîne	GPS/DestDistanceRef	Valeurs possibles : Texte (1 caractère)
GPS dest latitude	Chaîne	GPS/DestLatitude	Valeurs possibles : Texte

Constante	Type	Valeur	Commentaire
GPS dest latitude deg	Chaîne	GPS/DestLatitude/Deg	Valeurs possibles : Réel
GPS dest latitude dir	Chaîne	GPS/DestLatitude/Dir	Valeurs possibles : Texte (1 caractère)
GPS dest latitude min	Chaîne	GPS/DestLatitude/Min	Valeurs possibles : Réel
GPS dest latitude sec	Chaîne	GPS/DestLatitude/Sec	Valeurs possibles : Réel
GPS dest longitude	Chaîne	GPS/DestLongitude	Valeurs possibles : Texte
GPS dest longitude deg	Chaîne	GPS/DestLongitude/Deg	Valeurs possibles : Réel
GPS dest longitude dir	Chaîne	GPS/DestLongitude/Dir	Valeurs possibles : Texte (1 caractère)
GPS dest longitude min	Chaîne	GPS/DestLongitude/Min	Valeurs possibles : Réel
GPS dest longitude sec	Chaîne	GPS/DestLongitude/Sec	Valeurs possibles : Réel
GPS differential	Chaîne	GPS/Differential	Valeurs possibles : GPS Correction Applied, GPS Correction Not Applied
GPS DOP	Chaîne	GPS/DOP	Valeurs possibles : Réel
GPS img direction	Chaîne	GPS/ImgDirection	Valeurs possibles : GPS Magnetic north, GPS True north
GPS img direction ref	Chaîne	GPS/ImgDirectionRef	Valeurs possibles : GPS Magnetic north, GPS True north
GPS latitude	Chaîne	GPS/Latitude	Valeurs possibles : GPS North, GPS South
GPS latitude deg	Chaîne	GPS/Latitude/Deg	Valeurs possibles : Réel
GPS latitude dir	Chaîne	GPS/Latitude/Dir	Valeurs possibles : GPS North, GPS South
GPS latitude min	Chaîne	GPS/Latitude/Min	Valeurs possibles : Réel
GPS latitude sec	Chaîne	GPS/Latitude/Sec	Valeurs possibles : Réel
GPS longitude	Chaîne	GPS/Longitude	Valeurs possibles : GPS West, GPS East
GPS longitude deg	Chaîne	GPS/Longitude/Deg	Valeurs possibles : Réel
GPS longitude dir	Chaîne	GPS/Longitude/Dir	Valeurs possibles : GPS West, GPS East
GPS longitude min	Chaîne	GPS/Longitude/Min	Valeurs possibles : Réel
GPS longitude sec	Chaîne	GPS/Longitude/Sec	Valeurs possibles : Réel
GPS map date	Chaîne	GPS/MapDate	Valeurs possibles : Texte
GPS measure mode	Chaîne	GPS/MeasureMode	Valeurs possibles : GPS 2D, GPS 3D
GPS processing method	Chaîne	GPS/ProcessingMethod	Valeurs possibles : Texte
GPS satellites	Chaîne	GPS/Satellites	Valeurs possibles : Texte
GPS speed	Chaîne	GPS/Speed	Valeurs possibles : GPS km h, GPS miles h, GPS knots h
GPS speed ref	Chaîne	GPS/SpeedRef	Valeurs possibles : GPS km h, GPS miles h, GPS knots h
GPS status	Chaîne	GPS/Status	Valeurs possibles : GPS Measurement in progress, GPS Measurement Interoperability
GPS track	Chaîne	GPS/Track	Valeurs possibles : Réel (0.00..359.99)
GPS track ref	Chaîne	GPS/TrackRef	Valeurs possibles : Texte (1 caractère)
GPS version ID	Chaîne	GPS/VersionID	Valeurs possibles : Tableau Entier long (4 caractères)
IPTC byline	Chaîne	IPTC/Byline	Valeurs possibles : Texte ou Tableau Texte
IPTC byline title	Chaîne	IPTC/BylineTitle	Valeurs possibles : Texte ou Tableau Texte
IPTC caption abstract	Chaîne	IPTC/CaptionAbstract	Valeurs possibles : Texte
IPTC category	Chaîne	IPTC/Category	Valeurs possibles : Texte
IPTC city	Chaîne	IPTC/City	Valeurs possibles : Texte

Constante	Type	Valeur	Commentaire
IPTC contact	Chaîne	IPTC/Contact	Valeurs possibles : Texte ou Tableau Texte
IPTC content location code	Chaîne	IPTC/ContentLocationCode	Valeurs possibles : Texte ou Tableau Texte
IPTC content location name	Chaîne	IPTC/ContentLocationName	Valeurs possibles : Texte ou Tableau Texte
IPTC copyright notice	Chaîne	IPTC/CopyrightNotice	Valeurs possibles : Texte
IPTC country primary location code	Chaîne	IPTC/CountryPrimaryLocationCode	Valeurs possibles : Texte
IPTC country primary location name	Chaîne	IPTC/CountryPrimaryLocationName	Valeurs possibles : Texte
IPTC credit	Chaîne	IPTC/Credit	Valeurs possibles : Texte
IPTC date time created	Chaîne	IPTC/DateTimeCreated	Valeurs possibles : Date ou Texte (Datetime XML)
IPTC digital creation date time	Chaîne	IPTC/DigitalCreationDateTime	Valeurs possibles : Date ou Texte (Datetime XML)
IPTC edit status	Chaîne	IPTC/EditStatus	Valeurs possibles : Texte
IPTC expiration date time	Chaîne	IPTC/ExpirationDateTime	Valeurs possibles : Date ou Texte (Datetime XML)
IPTC fixture identifier	Chaîne	IPTC/FixtureIdentifier	Valeurs possibles : Texte
IPTC headline	Chaîne	IPTC/Headline	Valeurs possibles : Texte
IPTC image orientation	Chaîne	IPTC/ImageOrientation	Valeurs possibles : Texte
IPTC image type	Chaîne	IPTC/ImageType	Valeurs possibles : Texte
IPTC keywords	Chaîne	IPTC/Keywords	Valeurs possibles : Texte ou Tableau Texte
IPTC language identifier	Chaîne	IPTC/LanguageIdentifier	Valeurs possibles : Texte
IPTC object attribute reference	Chaîne	IPTC/ObjectAttributeReference	Valeurs possibles : Texte
IPTC object cycle	Chaîne	IPTC/ObjectCycle	Valeurs possibles : Texte
IPTC object name	Chaîne	IPTC/ObjektName	Valeurs possibles : Texte
IPTC original transmission reference	Chaîne	IPTC/OriginalTransmissionReference	Valeurs possibles : Texte
IPTC originating program	Chaîne	IPTC/OriginatingProgram	Valeurs possibles : Texte
IPTC program version	Chaîne	IPTC/ProgramVersion	Valeurs possibles : Texte
IPTC province state	Chaîne	IPTC/ProvinceState	Valeurs possibles : Texte
IPTC release date time	Chaîne	IPTC/ReleaseDateTime	Valeurs possibles : Date ou Texte (Datetime XML)
IPTC scene	Chaîne	IPTC/Scene	Valeurs possibles : IPTC Action, IPTC Aerial View, IPTC Close Up, IPTC Couple, IPTC Exterior View, IPTC Full Length, IPTC General View, IPTC Group, IPTC Half Length, IPTC Headshot, IPTC Interior View, IPTC Movie Scene, IPTC Night Scene, IPTC Off Beat, IPTC Panoramic View, IPTC Performing, IPTC Posing, IPTC Profile, IPTC Rear View, IPTC Satellite, IPTC Single, IPTC Symbolic, IPTC Two, IPTC Under Water
IPTC source	Chaîne	IPTC/Source	Valeurs possibles : Texte
IPTC special instructions	Chaîne	IPTC/SpecialInstructions	Valeurs possibles : Texte
IPTC star rating	Chaîne	IPTC/StarRating	Valeurs possibles : Entier long
IPTC sub location	Chaîne	IPTC/SubLocation	Valeurs possibles : Texte
IPTC subject reference	Chaîne	IPTC/SubjectReference	Valeurs possibles : Entier long ou Tableau Entier long
IPTC			

Constante	Type	Valeur	Comment
Supplemental category	Chaîne	IPTC/SupplementalCategory	Valeurs possibles : Texte ou Tableau Texte
IPTC urgency	Chaîne	IPTC/Urgency	Valeurs possibles : Entier long
IPTC writer editor	Chaîne	IPTC/WriterEditor	Valeurs possibles : Texte ou Tableau Texte
TIFF artist	Chaîne	TIFF/Artist	Valeurs possibles : Texte
TIFF compression	Chaîne	TIFF/Compression	Valeurs possibles : TIFF Adobe Deflate, TIFF CCIRLEW, TIFF CCITT1D, TIFF DCS, TIFF Deflate, TIFF Epson ERF, TIFF IT8BL, TIFF IT8CTPAD, TIFF IT8LW, TIFF IT8MP, TIFF JBIG, TIFF JBIGB&W, TIFF JBIGColor, TIFF JPEG, TIFF JPEG2000, TIFF JPEGThumbs Only, TIFF Kodak262, TIFF Kodak DCR, TIFF Kodak KDC, TIFF LZW, TIFF MDIBinary Level Codec, TIFF MDIProgressive Transform Codec, TIFF MDIVector, TIFF Next, TIFF Nikon NEF, TIFF Pack Bits, TIFF Pentax PEF, TIFF Pixar Film, TIFF Pixar Log, TIFF SGILog, TIFF SGILog24, TIFF Sony ARW, TIFF T4Group3Fax, TIFF T6Group4Fax, TIFF Thunderscan, TIFF Uncompressed
TIFF copyright	Chaîne	TIFF/Copyright	Valeurs possibles : Texte
TIFF date time	Chaîne	TIFF/DateTime	Valeurs possibles : Date ou Texte (Datetime XML)
TIFF document name	Chaîne	TIFF/DocumentName	Valeurs possibles : Texte
TIFF host computer	Chaîne	TIFF/HostComputer	Valeurs possibles : Texte
TIFF image description	Chaîne	TIFF/ImageDescription	Valeurs possibles : Texte
TIFF make	Chaîne	TIFF/Make	Valeurs possibles : Texte
TIFF model	Chaîne	TIFF/Model	Valeurs possibles : Texte
TIFF orientation	Chaîne	TIFF/Orientation	Valeurs possibles : TIFF Horizontal, TIFF Mirror Horizontal, TIFF Mirror Horizontal And Rotate270CW, TIFF Mirror Horizontal And Rotate90CW, TIFF Mirror Vertical, TIFF Rotate180, TIFF Rotate270CW, TIFF Rotate90CW
TIFF photometric interpretation	Chaîne	TIFF/PhotometricInterpretation	Valeurs possibles : TIFF Black Is Zero, TIFF CIELab, TIFF CMYK, TIFF Color Filter Array, TIFF ICCLab, TIFF ITULab, TIFF Linear Raw, TIFF Pixar Log L, TIFF Pixar Log Luv, TIFF RGB, TIFF RGBPalette, TIFF Transparency Mask, TIFF White Is Zero, TIFF YCb Cr
TIFF resolution unit	Chaîne	TIFF/ResolutionUnit	Valeurs possibles : TIFF CM, TIFF Inches, TIFF MM, TIFF None, TIFF UM
TIFF software	Chaîne	TIFF/Software	Valeurs possibles : Texte
TIFF xResolution	Chaîne	TIFF/XResolution	Valeurs possibles : Réel (lecture seulement)
TIFF yResolution	Chaîne	TIFF/YResolution	Valeurs possibles : Réel (lecture seulement)

Numéros de port TCP

Constante	Type	Valeur	Comment
TCP Authentication	Entier long	113	
TCP DNS	Entier long	53	
TCP Finger	Entier long	79	
TCP FTP Control	Entier long	21	
TCP FTP Data	Entier long	20	
TCP Gopher	Entier long	70	
TCP HTTP WWW	Entier long	80	
TCP IMAP3	Entier long	220	
TCP Kerberos	Entier long	88	
TCP KLogin	Entier long	543	
TCP Nickname	Entier long	43	
TCP NNTP	Entier long	119	
TCP NTalk	Entier long	518	
TCP NTP	Entier long	123	
TCP PMCP	Entier long	1643	
TCP PMD	Entier long	1642	
TCP POP3	Entier long	110	
TCP Printer	Entier long	515	
TCP RADACCT	Entier long	1646	
TCP RADIUS	Entier long	1645	
TCP Remote Cmd	Entier long	514	
TCP Remote Exec	Entier long	512	
TCP Remote Login	Entier long	513	
TCP Router	Entier long	520	
TCP SMTP	Entier long	25	
TCP SNMP	Entier long	161	
TCP SNMPTRAP	Entier long	162	
TCP SUN RPC	Entier long	111	
TCP Talk	Entier long	517	
TCP Telnet	Entier long	23	
TCP TFTP	Entier long	69	
TCP UUCP	Entier long	540	
TCP UUCP RLOGIN	Entier long	541	

Objets de formulaire (Accès)

Constante	Type	Valeur	Comment
Form hérité	Entier long	4	Retourne uniquement les objets hérités
Form page courante	Entier long	1	Retourne tous les objets de la page courante, y compris ceux de la page 0, mais exclut les objets hérités
Form toutes les pages	Entier long	2	Retourne tous les objets de toutes les pages, mais exclut les objets hérités
Objet avec focus	Entier long	1	
Objet conteneur sous formulaire	Entier long	2	
Objet courant	Entier long	0	
Objet nommé	Entier long	3	
Objet Premier ordre saisie	Chaîne	;FirstObject	

Objets de formulaire (Propriétés)

Constante	Type	Valeur	Comment
Activer événements autres inchangés	Entier long	1	Tous les événements listés dans le tableau *tabEvénements* sont activés, le statut des autres événements est inchangé
Activer événements inactiver autres	Entier long	0	Tous les événements listés dans le tableau *tabEvénements* sont activés, tous les autres événements sont désactivés
Aligné à droite	Entier long	4	
Aligné à gauche	Entier long	2	
Aligné au centre	Entier long	3	
Aligné en bas	Entier long	4	
Aligné en haut	Entier long	2	
Aligné par défaut	Entier long	1	
Bordure Aucune	Entier long	0	Les objets apparaissent sans encadrement
Bordure Double	Entier long	5	Les objets apparaissent encadrés par une double ligne, c'est-à-dire deux lignes continues de 1 pt séparées par un pixel
Bordure Normal	Entier long	1	Les objets apparaissent dans un cadre continu de 1 pt
Bordure Relief	Entier long	3	Les objets apparaissent avec un effet 3D (relief)
Bordure Relief inversé	Entier long	4	Les objets apparaissent avec un effet 3D en creux (relief inversé)
Bordure Système	Entier long	6	Le cadre est dessiné en fonction des spécifications graphiques du système
Bordure Trait pointillé	Entier long	2	Les objets apparaissent dans un cadre pointillé de 1 pt
Impression limitée avec report	Entier long	2	La taille initiale de la zone du sous-formulaire est conservée mais 4D imprime le formulaire plusieurs fois pour imprimer tous les enregistrements.
Impression limitée par le cadre	Entier long	1	4D n'imprime que les enregistrements qui apparaissent dans la zone du sous-formulaire. Le formulaire n'est imprimé qu'une fois et les enregistrements qui ne sont pas imprimés sont ignorés.
Inactiver événements autres inchangés	Entier long	2	Tous les événements listés dans le tableau *tabEvénements* sont désactivés, le statut des autres événements est inchangé
Indicateur Barber shop	Entier long	2	Barre affichant une animation continue
Indicateur Barre de progression	Entier long	1	Barre de progression standard
Indicateur de progression asynchrone	Entier long	3	Indicateur circulaire affichant une animation continue
Liste énumération	Entier long	0	Liste simple de choix de valeurs (option "Enumération" dans la Liste des propriétés) (défaut)
Liste exclusions	Entier long	2	Liste de valeurs non acceptées pour la saisie (option "Exclusions" dans la Liste des propriétés)
Liste obligations	Entier long	1	Liste des seules valeurs acceptées pour la saisie (option "Obligations" dans la Liste des propriétés)
Multiligne Auto	Entier long	0	Dans les zones mono-lignes, les mots situés en fin de ligne sont tronqués et il n'y a pas de retours à la ligne. Dans les zones multi-lignes, 4D effectue des retours à la ligne automatiques
Multiligne Non	Entier long	2	Il n'y a aucun retour à la ligne : le texte est toujours affiché sur une seule ligne. Si le champ ou la variable alpha ou texte contient des retour chariots, le texte situé après le premier retour chariot est effacé dès que la zone est modifié
Multiligne Oui	Entier long	1	Dans les zones mono-lignes, le texte est affiché jusqu'au premier retour chariot ou au dernier mot affichable en entier. 4D insère des retours à la ligne, il est possible de faire défiler le contenu de la zone en appuyant sur la touche flèche basse. Dans les zones multi-lignes, 4D effectue des retours à la ligne automatiques
Orientation 0°	Entier long	0	Pas de rotation (valeur par défaut)
Orientation 180°	Entier long	180	Orientation du texte à 180° dans le sens horaire
Orientation 90° droite	Entier long	90	Orientation du texte à 90° dans le sens horaire
Orientation 90° gauche	Entier long	270	Orientation du texte à 90° dans le sens anti-horaire
Redim horizontal agrandir	Entier long	1	Si la fenêtre s'agrandit de 50% en largeur, l'objet s'élargit de 50% vers la droite
Redim horizontal aucun	Entier long	0	Si la fenêtre s'agrandit en largeur, ni la largeur ni la position de l'objet ne varient
Redim horizontal déplacer	Entier long	2	Si la fenêtre s'agrandit de 100 pixels en largeur, l'objet se déplace de 100 pixels vers la droite

Constante	Type	Valeur	Comment
Redim vertical agrandir	Entier long	1	Si la fenêtre s'agrandit de 50% en hauteur, l'objet s'allonge de 50% vers le bas
Redim vertical aucun	Entier long	0	Si la fenêtre s'agrandit en hauteur, ni la hauteur ni la position de l'objet ne varient
Redim vertical déplacer	Entier long	2	Si la fenêtre s'agrandit de 100 pixels en hauteur, l'objet se déplace de 100 pixels vers le bas

Options d'impression

Constante	Type	Valeur	Comment
Dialogue d'impression	Entier long	2	Affichage de la boîte de dialogue d'impression
Dialogue de format d'impression	Entier long	1	Affichage de la boîte de dialogue Format d'impression
Driver PDF générique	Chaîne	_4d_pdf_printer	**Note :** Cette fonctionnalité n'est pas disponible dans les versions 32 bits de 4D. • Sous OS X, déclare le pilote par défaut comme imprimante courante. Ce pilote n'est pas visible et ne se trouve pas dans la liste retournée par la commande **LISTE IMPRIMANTES**. Notez que le chemin d'accès pour le document PDF doit être défini en utilisant la commande **FIXER OPTION IMPRESSION**, sinon l'erreur 3107 est retournée. • Sous Windows, déclare le pilote PDF de Windows (nommé "Microsoft Print to PDF") comme imprimante courante. Cette constante est disponible sous Windows 10 uniquement, lorsque l'option PDF est installée. Avec d'autres versions de Windows, ou lorsqu'il n'y a pas de pilote PDF disponible, la commande ne fait rien et la variable système *OK* prend la valeur 0.
Option alimentation	Entier long	5	(Windows uniqument) *valeur1* uniquement : numéro correspondant à l'indice, dans le tableau des bacs retourné par la commande **VALEURS OPTION IMPRESSION**, du bac papier à utiliser. Cette option est utilisable sous Windows uniquement.
Option ancienne couche impression	Entier long	16	(Versions 4D 64 bits pour Windows uniquement) *valeur1* uniquement : 1=sélectionner l'ancienne couche d'impression GDI pour toutes les tâches d'impression suivantes, 0=sélectionner la couche d'impression D2D (défaut). **Versions 64 bits :** Ce sélecteur est pris en charge dans les applications 4D 64 bits monopostes sous Windows uniquement, et est ignoré pour les autres plates-formes. Il est principalement destiné, dans ces applications, à permettre aux plug-ins d'ancienne génération d'imprimer dans des tâches d'impression 4D.
Option couleur	Entier long	8	(Windows uniquement) *valeur1* uniquement : code indiquant le mode de prise en charge de la couleur : 1=Noir et blanc (monochrome), 2=Couleur. **Versions 64 bits** : Cette option n'est pas prise en charge dans les versions 64 bits de 4D (obsolète).
Option destination	Entier long	9	*valeur1*: code indiquant le type de destination de l'impression : 1=Imprimante, 2=Fichier (PC)/PS (Mac), 3=Fichier PDF, 5=Ecran (option du pilote OS X) Si *valeur1* est différent de 1 ou de 5, *valeur2* contient un chemin d'accès pour le document résultant. Ce chemin sera utilisé jusqu'à ce qu'un autre chemin soit spécifié. Si un fichier du même nom existe déjà à l'emplacement de destination, il est remplacé. Avec **LIRE OPTION IMPRESSION**, si la valeur courante n'est pas dans la liste prédéfinie, *valeur1* contient -1 et la variable système OK vaut 1. Si une erreur se produit, *valeur1* et la variable système OK valent 0. **Note :** Sous Windows, vous pouvez définir la destination d'impression 3 (Fichier PDF) lorsque le pilote PDF Creator a été installé. Lorsque les valeurs (*9;3;chemin*) sont passées, 4D lance automatiquement une impression PDF "silencieuse" et prend en compte les codes d'options éventuellement passés (à noter que si vous passez une chaîne vide dans *valeur2* ou omettez ce paramètre, une boîte de dialogue d'enregistrement de fichier apparaît au moment de l'impression). A l'issue de l'impression, les paramétrages courants sont restaurés. Ce principe simplifie le pilotage des impressions PDF par 4D et permet d'écrire du code multi-plate-forme. Si les valeurs (*9;3;chemin*) ne sont pas passées, l'impression n'est pas pilotée par 4D et les éventuels codes d'options de PDF Creator sont ignorés.
Option échelle	Entier long	3	*valeur1* uniquement : valeur d'échelle en pourcentage. Attention, certaines imprimantes ne permettent pas de modifier l'échelle. Si vous passez une valeur invalide, la propriété est remise à 100% au moment de l'impression.
Option intervalle de page	Entier long	15	*valeur1*=numéro de la première page à imprimer (valeur par défaut 1) et (optionnel) *valeur2*=numéro de la dernière page à imprimer (valeur par défaut -1 = fin du document).
Option masquer progression impr	Entier long	14	*valeur1* uniquement : 1=masquer toutes les fenêtres de progression d'impression, 0=afficher les fenêtres de progression d'impression (fonctionnement par défaut). Cette option est particulièrement utile dans le cadre des impressions en PDF sous OS X. **Note :** Il existe déjà une option d'affichage Progression de l'impression accessible via la boîte de dialogue des Propriétés de la base (page Interface). Toutefois, elle est globale à l'application et ne masque pas toutes les fenêtres sous OS X.
Option mode impression Mac	Entier long	13	(Mac uniquement) *valeur1* uniquement : 0=impression en mode PDF (valeur par défaut), 1=impression en mode Postscript. **Notes :** - Cette option n'a pas d'effet sous Windows. - Sous OS X, les impressions sont effectuées par défaut en mode PDF. Or, le pilote d'impression PDF ne prend pas en charge les images PICT encapsulant des informations Postscript — ces images sont générées notamment par des logiciels de dessin vectoriel. Pour résoudre ce problème, cette option permet de modifier le mode d'impression sous OS X pour la session courante. Attention, l'impression en mode Postscript peut entraîner des effets de bords indésirables. **Versions 64 bits :** Cette option n'est pas prise en charge ; elle est remplacée par l'utilisation de l'option Driver PDF générique de la commande **FIXER IMPRIMANTE COURANTE**.
Option nom document à imprimer	Entier long	12	*valeur1* uniquement : nom du document d'impression, qui apparaît dans la liste des documents du serveur d'impression. Le nom défini par cette instruction sera utilisé pour tous les documents d'impression de la session tant qu'un nouveau nom ou une chaîne vide ne sera pas passé(e). Pour utiliser ou rétablir le fonctionnement standard (utilisation du nom de la méthode dans le cas d'une méthode, nom de la table pour un enregistrement, etc.), passez une chaîne vide dans *valeur1*.
Option nombre copies	Entier long	4	*valeur1* uniquement : nombre de copies à imprimer
Option orientation	Entier long	2	*valeur1* uniquement : 1=Portrait, 2=Paysage. Si une option d'orientation différente est utilisée, **LIRE OPTION IMPRESSION** retourne 0 dans *valeur1*. **Versions 64 bits** : Cette option peut être appelée au sein d'une tâche d'impression, ce qui signifie que vous

Constante	Type	Valeur	Comment
			Pouvez passer du mode portrait au mode paysage et inversement dans la même tâche d'impression.
Option papier	Entier long	1	Si vous passez uniquement *valeur1*, il contient le nom du papier. Si vous passez les deux paramètres, *valeur1* contient la largeur du papier et *valeur1* contient la hauteur du papier. La largeur et la hauteur sont exprimées en pixels écran. Utilisez la commande **VALEURS OPTION IMPRESSION** pour connaître le nom, la hauteur et la largeur de tous les formats de papier proposés par l'imprimante.
Option recto verso	Entier long	11	(Windows uniquement) *valeur1*: 0=Recto ou standard, 1=Recto-verso. Si *valeur1*=1, *valeur2* contient la reliure à appliquer : 0=Reliure à gauche (valeur par défaut), 1=Reliure en haut. **Note :** Cette option est utilisable sous Windows uniquement.
PDFCreator Nom imprimante	Chaîne	PDFCreator	

Paramètre des graphes

Constante	Type	Valeur	Comment
Graphe afficher la légende	Chaîne	displayLegend	**Valeurs possibles** : Booléen **Valeur par défaut** : Vrai
Graphe couleur fond	Chaîne	graphBackgroundColor	**Valeurs possibles :** Expression couleur norme SVG (texte), par exemple "#7F8E00", "Pink" ou "#0a1414"
Graphe couleur fond document	Chaîne	documentBackgroundColor	**Valeurs possibles** : Expression couleur SVG (texte), par exemple "#7F8E00", "Pink" ou "#0a1414". Lorsqu'un graphe enregistré sous forme d'image SVG est ouvert dans une autre application, la couleur de fond du document est affichée uniquement si le moteur de rendu SVG prend en charge la norme *SVG tiny 1.2* (prise en charge sur IE, Firefox mais pas sur Chrome).
Graphe couleur ombre	Chaîne	graphBackgroundShadowColor	**Valeurs possibles** : Expression couleur SVG (texte), par exemple "#7F8E00", "Pink" ou "#0a1414"
Graphe couleur police	Chaîne	fontColor	**Valeurs possibles** : Expression couleur SVG (texte), par exemple "#7F8E00", "Pink" ou "#0a1414".
Graphe couleur police légende	Chaîne	legendFontColor	**Valeurs possibles** : Expression couleur SVG (texte), par exemple "#7F8E00", "Pink" ou "#0a1414"
Graphe couleurs	Chaîne	colors	**Valeurs possibles** : Tableau texte. Couleurs pour chaque série de graphe. **Valeurs par défaut** : Blue-green (#19BAC9), Yellow (#FFC338), Purple (#573E82), Green (#4FA839), Orange (#D95700), Blue (#1D9DF2), Yellow-green (#B5CF32), Red (#D43A26)
Graphe décalage secteurs	Chaîne	pieShift	**Valeurs possibles** : Réels **Valeur par défaut** : 8
Graphe épaisseur des lignes	Chaîne	lineWidth	**Valeurs possibles** : Réels **Valeur par défaut** : 2
Graphe espacement colonnes	Chaîne	columnGap	**Valeurs possibles** : Entier long **Valeur par défaut** : 12 Définit l'espacement entre les colonnes du graphe
Graphe espacement icônes légende	Chaîne	legendIconGap	**Valeurs possibles** : Réel **Valeur par défaut** : Graphe hauteur icônes légende/2
Graphe hauteur des points	Chaîne	plotHeight	**Valeurs possibles** : Réels **Valeur par défaut** : 12
Graphe hauteur icônes légende	Chaîne	legendIconHeight	**Valeurs possibles** : Réels **Valeur par défaut** : 20
Graphe hauteur par défaut	Chaîne	defaultHeight	**Valeurs possibles** : Réel **Valeur par défaut** : 400. Si graphType=7 (secteurs), valeur défaut = 600
Graphe largeur des points	Chaîne	plotWidth	**Valeurs possibles** : Réel **Valeur par défaut** : 12
Graphe largeur icônes légende	Chaîne	legendIconWidth	**Valeurs possibles** : Réel **Valeur par défaut** : 20
Graphe largeur max colonne	Chaîne	columnWidthMax	**Valeurs possibles** : Réel **Valeur par défaut** : 200
Graphe largeur min colonne	Chaîne	columnWidthMin	**Valeurs possibles** : Réel **Valeur par défaut** : 10
Graphe largeur par défaut	Chaîne	defaultWidth	**Valeurs possibles** : Réel **Valeur par défaut** : 600. Si graphType=7 (Secteurs), valeur par défaut = 800
Graphe libellés légende	Chaîne	legendLabels	**Valeurs possibles** : Tableau texte. S'il est manquant, 4D affiche des icônes sans texte.
Graphe marge basse	Chaîne	bottomMargin	**Valeurs possibles** : Nombre réel **Valeur par défaut** : 12
Graphe marge droite	Chaîne	rightMargin	**Valeurs possibles** : Réels **Valeur par défaut** : 2
Graphe marge gauche	Chaîne	leftMargin	**Valeurs possibles** : Réel **Valeur par défaut** : 12
Graphe			**Valeurs possibles** : Réels

Constante	Type	Valeurs	Comment
Graphe haute	Chaîne	topMargin	**Valeur par défaut** : 2
Graphe opacité fond	Chaîne	graphBackgroundOpacity	**Valeurs possibles** : Entier long entre 0 et 100 **Valeur par défaut** : 100
Graphe opacité fond document	Chaîne	documentBackgroundOpacity	**Valeurs possibles** : Entier long (0 à 100). Lorsqu'un graphe enregistré sous forme d'image SVG est ouvert dans une autre application, l'opacité du fond du document est affichée uniquement si le moteur de rendu SVG prend en charge la norme *SVG tiny 1.2* (prise en charge sur IE, Firefox mais pas sur Chrome). **Valeur par défaut :** 100
Graphe rayon des points	Chaîne	plotRadius	**Valeurs possibles** : Réels **Valeur par défaut** : 12
Graphe taille police	Chaîne	fontSize	**Valeurs possibles** : Entier long **Valeur par défaut** : 12. Si graphType=7 (Secteurs), voir Graphe taille police des secteurs
Graphe taille police des secteurs	Chaîne	pieFontSize	**Valeurs possibles** : Réels **Valeur par défaut** : 16
Graphe type	Chaîne	graphType	**Valeurs possibles** : Entier long [1 à 8] où 1 = colonnes, 2 = colonnes proportionnelles, 3 = colonnes empilées, 4 = lignes, 5 = aires, 6 = points, 7 = secteurs, 8 = images. **Valeur par défaut** : 1 Si la valeur est nulle, le graphe n'est pas dessiné et aucun message d'erreur n'est affiché. Si la valeur est trop grande, le graphe n'est pas dessiné et un message d'erreur est affiché. Si vous souhaitez modifier les graphes de type image (valeur=8), vous devez recopier le dossier 4D/Resources/GraphTemplates/Graph_8_Pictures/ dans le dossier Resources de votre base et effectuer les modifications nécessaires. Les fichiers image locaux seront utilisés au lieu des fichiers de 4D. Les noms des fichiers image sont libres ; 4D trie les fichiers contenus dans le dossier et affecte le premier fichier au premier graphe. Ces fichiers peuvent être de type SVG ou image.
Graphe xGrille	Chaîne	xGrid	**Valeurs possibles** : Booléen **Valeur par défaut** : Vrai Utilisé uniquement avec les types proportionnels 4 et 6
Graphe xMax	Chaîne	xMax	**Valeurs possibles** : Nombre, Date, Heure (même type que le paramètre *xCatégories*). Seules les valeurs inférieures à xMax sont affichées dans le graphe. xMax est utilisé uniquement pour les graphes de type 4, 5, or 6 si xProp=Vrai et si *xCatégories* est de type numérique, date ou heure. Si ce paramètre est manquant ou si xMin>xMax, 4D calcule automatiquement la valeur xMax.
Graphe xMin	Chaîne	xMin	**Valeurs possibles** : Nombre, Date, Heure (même type que le paramètre *xCatégories*). Seules les valeurs supérieures xMin sont affichées dans le graphe. xMin est utilisé uniquement pour les graphes de type 4, 5, or 6 si xProp=Vrai et si *xCatégories* est de type numérique, date ou heure. Si ce paramètre est manquant ou si xMin>xMax, 4D calcule automatiquement la valeur xMin.
Graphe xProp	Chaîne	xProp	**Valeurs possibles** : Booléen **Valeur par défaut** : Faux Vrai pour un axe x proportionnel, Faux pour un axe x normal. Utilisé uniquement avec les types proportionnels 4, 5 et 6
Graphe yGrille	Chaîne	yGrid	**Valeurs possibles** : Booléen **Valeur par défaut** : Vrai
Graphe yMax	Chaîne	yMax	**Valeurs possibles** : Nombres Si ce paramètre est manquant, 4D calcule automatiquement la valeur yMax.
Graphe yMin	Chaîne	yMin	**Valeurs possibles** : Nombres Si ce paramètre est manquant, 4D calcule automatiquement la valeur yMin.

Paramètres de formulaire

Constante	Type	Valeur	Comment
Objets non inversés	Entier long	0	
Pas de sélection	Entier long	0	Il n'est pas possible de sélectionner un enregistrement dans la liste
Sélection multiple	Entier long	2	L'utilisateur peut sélectionner plusieurs enregistrements. Pour sélectionner des enregistrements contigus, il suffit de cliquer sur le premier enregistrement à sélectionner puis d'appuyer sur la touche **Majuscule** avant de cliquer sur le dernier. Pour sélectionner des enregistrements non adjacents, il suffit de cliquer sur chaque enregistrement en maintenant enfoncée la touche **Ctrl** (sous Windows) ou **Commande** (sous Mac OS).
Sélection unique	Entier long	1	Seule la sélection d'un enregistrement à la fois est autorisée

Paramètres de la base

Constante	Type	Valeur	Comment
_o_Enreg requêtes Web	Entier long	29	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
_o_Mode conversion Web	Entier long	8	**** *Sélecteur inactivé* ****
_o_Précision affichage réels	Entier long	32	**** *Sélecteur inactivé* ****
_o_Taille cache données	Entier long	9	**Portée** : Application 4D **Conservé entre deux sessions** : - **Description** : *Constante obsolète (conservée uniquement pour des raisons de compatibilité).* L'utilisation de la commande **Lire taille cache** est désormais recommandée.
Adresse IP d'écoute	Entier long	16	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Appels système 4D mode distant	Entier long	12	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Valeurs possibles** : pour les sélecteurs 10, 11 et 12, le paramètre *valeur* est de la forme hexadécimale *0x00aabbcc* dans laquelle : *aa* = nombre minimum de ticks par appel au système (de 0 à 100 inclus) *bb* = nombre maximum de ticks par appel au système (de 0 à 100 inclus) *cc* = nombre de ticks entre deux appels au système (de 0 à 20 inclus). Si une des valeurs est hors de son intervalle, 4D la fixe à son maximum. Vous pouvez également passer dans *valeur* une des trois valeurs suivantes, correspondant à un ensemble de réglages standard : • *valeur* = -1 : priorité maximum allouée à 4D, • *valeur* = -2 : priorité moyenne allouée à 4D, • *valeur* = -3 : priorité minimum allouée à 4D. **Description** : Ce sélecteur vous permet de fixer dynamiquement les réglages des appels système de 4D. En fonction du Sélecteur, le gestionnaire d'appels système sera défini pour : • 4D mode local lorsque la commande est appelée depuis 4D monoposte (*sélecteur=10*). • 4D Server lorsque la commande est appelée depuis 4D Server (*sélecteur=11*). • 4D mode distant lorsque la commande est appelée depuis 4D connecté à 4D Server (*sélecteur=12*). **Note :** Le fonctionnement du sélecteur 12 (Appels système 4D mode distant) diffère suivant que la commande **FIXER PARAMETRE BASE** est exécutée sur le poste serveur ou sur un poste client : - si la commande est exécutée sur le poste serveur, la nouvelle valeur sera appliquée à tous les postes clients se connectant ultérieurement. - si la commande est exécutée sur un poste client, la nouvelle valeur est appliquée immédiatement au poste client ainsi qu'à tous les postes clients se connectant par la suite au serveur. Vous pouvez utiliser ce fonctionnement pour mettre en place une gestion dynamique et individualisée de la priorité pour chaque poste client. Le principe consiste à exécuter la commande une première fois sur le poste client à configurer puis une seconde fois sur le poste serveur avec la valeur par défaut, qui sera alors utilisée pour les postes client se connectant par la suite. Ce fonctionnement est effectif dans 4D à partir des versions 6.8.6, 2003.3 et 2004. **Attention** : Paramétrer ces sélecteurs de façon inappropriée peut conduire à une forte dégradation des performances de l'application. Il est conseillé de ne modifier les valeurs par défaut qu'en parfaite connaissance de cause.
Appels système 4D mode local	Entier long	10	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Description** : voir sélecteur 12
Appels système 4D Server	Entier long	11	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Description** : voir sélecteur 12
Casse caractères moteur SQL	Entier long	44	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (casse non prise en compte) ou 1 (casse prise en compte) **Description** : Activation ou désactivation de la prise en compte de la casse des caractères pour les comparaisons de chaînes effectuées par le moteur SQL. Par défaut, la valeur est 1 (casse prise en compte) : le moteur SQL établit une différence entre les majuscules et les minuscules ainsi qu'entre les caractères accentués lors des comparaisons de chaînes (tris et recherches). Par exemple "ABC" = "ABC" mais "ABC" # "Abc" et "abc" # "âbc" . Dans certains cas, par exemple pour aligner le fonctionnement du moteur SQL sur celui du moteur 4D, vous pourrez souhaiter que les comparaisons de chaînes ne tiennent pas compte de la casse ("ABC"="Abc"="âbc"). Cette option peut également être définie dans la **Page SQL** des Propriétés de la base.
			Portée : Table et process courants **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (utiliser le paramétrage de la base), 1 (exécuter sur le client) ou 2 (exécuter sur le serveur) **Description** : Emplacement de l'exécution des commandes **CHERCHER PAR FORMULE** et **CHERCHER PAR FORMULE DANS SELECTION** pour la *table* passée en paramètre. Dans le cadre de l'exploitation d'une base en client-serveur, les commandes de recherche "par formule" peuvent exécutées soit sur le serveur soit sur le client :

Constante	Type	Valeur	Comment
Chercher par formule serveur	Entier long	46	• dans les bases de données créées à partir de 4D v11 SQL, ces commandes sont exécutées sur le serveur. • dans les bases de données converties, ces commandes sont exécutées sur le client, comme dans les versions précédentes de 4D. • dans les bases de données converties, une préférence spécifique permet de modifier globalement le lieu d'exécution de ces commandes. Cette différence de lieu d'exécution influe sur les performances de l'application (l'exécution sur le serveur est généralement plus rapide) mais également sur la programmation. En effet, la valeur des composantes de la formule (notamment les variables appelées via une méthode) diffère suivant le contexte d'exécution. Vous pouvez utiliser ce sélecteur pour adapter ponctuellement le fonctionnement de votre application. Si vous passez 0 dans le paramètre *valeur*, l'emplacement d'exécution des commandes de recherche "par formule" dépendra de la configuration de la base : dans les bases créées avec 4D v11 SQL, les commandes seront exécutées sur le serveur. Dans les bases converties, elles seront exécutées sur le client ou le serveur en fonction des préférences de la base. Passez 1 ou 2 dans *valeur* pour "forcer" l'exécution des commandes respectivement sur le client ou sur le serveur. Reportez-vous à l'exemple 4. **Note :** Si vous souhaitez pouvoir activer les jointures "type SQL" (cf. sélecteur Jointures chercher par formule), vous devez toujours exécuter les formules sur le serveur afin qu'elle ait accès aux enregistrements. Attention, dans ce contexte, la formule ne doit pas contenir d'appel à une méthode, sinon elle est automatiquement basculée sur le poste distant.
Client adresse IP d'écoute	Entier long	23	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 16 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Propriétés de la base 4D en mode distant.
Client enreg requêtes Web	Entier long	30	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 = Ne pas enregistrer (défaut), 1 = Enregistrer au format CLF, 2 = Enregistrer au format DLF, 3 = Enregistrer au format ELF, 4 = Enregistrer au format WLF. **Description** : Démarrage ou arrêt de l'enregistrement des requêtes Web reçues par les serveurs Web de tous les postes clients. Par défaut, la valeur est 0 (pas d'enregistrement des requêtes). Le fonctionnement de ce sélecteur est identique à celui du sélecteur 29 ; il s'applique toutefois à tous les postes 4D clients utilisés en tant que serveurs Web. Le fichier "logweb.txt" est dans ce cas automatiquement placé dans le sous-dossier Logs du dossier base 4D client (dossier de cache). Si vous souhaitez définir des valeurs pour certains postes clients uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client jeu de caractères	Entier long	24	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 17 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client maximum process Web	Entier long	20	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 7 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client minimum process Web	Entier long	19	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 6 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client numéro de port	Entier long	22	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 15 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client numéro de port HTTPS	Entier long	40	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 à 65535 **Description** : Numéro du port TCP utilisé par les serveurs Web des postes clients pour les connexions sécurisées via SSL (protocole HTTPS). Par défaut, la valeur est 443 (valeur standard). Le fonctionnement de ce sélecteur est identique à celui du sélecteur 39 ; il s'applique toutefois à tous les postes 4D distants utilisés en tant que serveurs Web. Si vous souhaitez modifier la valeur de certains postes clients uniquement, utilisez la boîte de dialogue des Préférences de 4D distant.
Client proc Web simultanés maxi	Entier long	25	**Portée** : Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 18 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.

Constante	Type	Valeur	Commentaire
			Portée: Tous postes 4D distants **Conservé entre deux sessions** : Oui **Valeurs possibles** : Voir sélecteur 27 **Description** : Permet de spécifier ce paramètre pour les postes 4D distants utilisés en tant que serveurs Web. La valeur définie via ce sélecteur est appliquée à tous les postes distants utilisés comme serveurs Web. Si vous souhaitez définir cette valeur pour certains postes distants uniquement, utilisez la boîte de dialogue des Préférences de 4D en mode distant.
Client taille max requêtes Web	Entier long	21	
Correcteur orthographique	Entier long	81	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (défaut) = correcteur OS X natif (Hunspell désactivé), 1 = correcteur Hunspell actif. **Description** : Permet d'activer le correcteur orthographique Hunspell sous OS X. Par défaut, sur cette plate-forme le correcteur natif est activé. Vous pouvez souhaiter utiliser le correcteur Hunspell par exemple pour unifier l'interface de vos applications multiplates-formes (sous Windows, seul le correcteur Hunspell est disponible). Pour plus d'informations, reportez-vous au paragraphe **Prise en charge des dictionnaires Hunspell**.
Direct2D désactivé	Entier long	0	Voir sélecteur 69 (Direct2D Statut)
Direct2D lire statut actif	Entier long	74	**Note :** Ce sélecteur peut être utilisé uniquement avec la commande **Lire parametre base**, sa valeur ne peut pas être fixée. **Description** : Retourne l'implémentation active de Direct2D sous Windows. **Valeurs possibles** : 0, 1, 2, 3, 4 ou 5 (cf. valeurs du sélecteur 69). La valeur retournée dépend de la disponibilité de Direct2D, du matériel et de la qualité de la prise en charge de Direct2D par le système d'exploitation. Par exemple, si vous exécutez : `FIXER PARAMETRE BASE(Direct2D Statut;Direct2D Matériel)` `$mode:=Lire parametre base(Direct2D Lire statut actif)` - sur Windows 7 et suivants, *$mode* vaudra 1 si le système détecte un matériel compatible Direct2D, sinon *$mode* vaudra 3 (contexte logiciel). - sur Windows Vista, *$mode* vaudra 1 si le système détecte un matériel compatible Direct2D, sinon *$mode* vaudra 0 (désactivation de Direct2D). - sur Windows XP, *$mode* vaudra toujours 0 (incompatibilité avec Direct2D).
Direct2D Logiciel	Entier long	3	Voir sélecteur 69 (Direct2D Statut)
Direct2D matériel	Entier long	1	Voir sélecteur 69 (Direct2D Statut)
Direct2D statut	Entier long	69	**Portée**: Application 4D **Conservé entre deux sessions** : Non **Description** : Mode d'activation de l'implémentation de Direct2D sous Windows. **Valeurs possibles** : Une des constantes suivantes (mode 3 par défaut) : Direct2D désactivé (0) : le mode Direct2D n'est pas activé, la base fonctionne dans le mode précédent (GDI/GDIPlus). Direct2D matériel (1) : utilisation de Direct2D en contexte graphique matériel dans toute l'application 4D. Si ce contexte n'est pas disponible, utilisation du contexte graphique Direct2D logiciel (hormis sous Vista, pour des raisons de performances dans ce cas le mode GDI/GDIPlus est utilisé). Direct2D Logiciel (3) (Mode par défaut) : à partir de Windows 7, utilisation de Direct2D en contexte graphique logiciel dans toute l'application 4D. Sous Vista, pour des raisons de performances le mode GDI/GDIPlus est utilisé. **Note de compatibilité :** A compter de 4D v14, les modes hybrides sont désactivés et sont redirigés vers les modes disponibles (l'ancien mode 2 équivaut au mode 1 ; les anciens modes 4 et 5 sont équivalents au mode 3).
Enreg diagnostic	Entier long	79	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 ou 1 (0 = ne pas enregistrer, 1 = enregistrer) **Description** : Démarrage ou arrêt de l'enregistrement du fichier de diagnostic de 4D. Par défaut, la valeur est 0 (pas d'enregistrement). 4D vous permet d'enregistrer de manière continue dans un fichier de diagnostic un ensemble d'événements relatifs au fonctionnement interne de l'application. Les informations contenues dans ce fichier sont destinées à la mise au point des applications 4D et pourront être analysées avec l'aide des services techniques de 4D. Lorsque vous passez 1 dans ce sélecteur, le fichier de diagnostic, nommé *NomBase*.txt, est automatiquement créé (ou ouvert) dans le dossier **Logs** de la base. Une fois que le fichier atteint une taille de 10 Mo, il est refermé et un nouveau fichier *NomBase*_N.txt est généré, avec un numéro séquentiel N incrémenté. A noter qu'il est possible d'inclure des informations personnalisées dans ce fichier à l'aide de la commande **ENREGISTRER EVENEMENT**.
Enreg événements debogage	Entier long	34	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Description** : Démarrage ou arrêt de l'enregistrement séquentiel des événements de programmation de 4D dans le fichier *4DDebugLog*, automatiquement placé dans le sous-dossier Logs de la base, à côté du fichier de structure. Un nouveau format texte tabulé, plus compact, est utilisable pour le fichier d'enregistrement des événements "4DDebugLog[_n].txt" à compter de 4D v14 (_n est le numéro de segment du fichier). **Valeurs possibles** : Entier long contenant un champ de bits (bit field) : valeur = bit1(1)+bit2(2)+bit3(4)+bit4(8)+…). - Le bit 1 (valeur 1) permet de demander à activer le fichier (à noter que toute autre valeur non nulle l'activera également) - Le bit 2 (valeur 2) permet de demander les paramètres d'appel aux méthodes et commandes. - Le bit 3 (valeur 4) permet d'activer le nouveau format tabulé. - Le bit 4 (valeur 8) permet de désactiver l'écriture immédiate de chaque opération sur disque (activée par défaut). L'écriture immédiate est moins rapide mais plus efficace par exemple pour rechercher les causes d'un plantage. Si vous désactivez ce mode, le contenu fichier sera plus compact et il sera généré plus rapidement. - Le bit 5 (valeur 16) permet de désactiver l'enregistrement des appels de plug-ins (activé par défaut). Dans le format non tabulé (ancien format), les durées d'exécution sont exprimées en millisecondes, la valeur "< ms" est affichée si une opération s'exécute en moins d'une milliseconde. Dans le nouveau format tabulé, les durées d'exécution sont exprimées en microsecondes. Exemples : FIXER PARAMETRE BASE(34;1) // active le fichier mode v13 sans les paramètres, avec les durées

Constante	Type	Valeur	Comment
Enreg requêtes 4D Server	Entier long	28	FIXER PARAMETRE BASE(34;2) // active le fichier mode v13 avec les paramètres et les durées FIXER PARAMETRE BASE(34;2+4) // active le fichier au format v14 avec les paramètres et les durées FIXER PARAMETRE BASE(34;0) // désactive le fichier Afin d'éviter que le fichier n'enregistre une trop grande quantité d'informations, vous pouvez restreindre les commandes 4D à examiner à l'aide du sélecteur 80, Liste commandes enreg. L'option peut être activée dans tout type d'application 4D (4D tous modes, 4D Server, 4D Volume Desktop), en interprété ou en compilé. **Note :** Cette option est proposée uniquement à des fins de débogage, elle ne doit pas être utilisée en production car elle peut entraîner une dégradation des performances de l'application ainsi que la saturation du disque dur. Pour plus d'informations sur le format et l'exploitation du fichier 4DDebugLog[_n].txt, veuillez contacter les services techniques de 4D SAS. **Portée** : *4D Server, 4D distant* **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 ou de 1 à N (0 = ne pas enregistrer, 1 à N = numéro séquentiel, accolé au nom du fichier). **Description** : Démarrage ou arrêt de l'enregistrement des requêtes standard reçues par 4D Server (hors requêtes Web). Par défaut, la valeur est 0 (pas d'enregistrement de requêtes). 4D Server vous permet d'enregistrer dans un fichier d'historique chaque requête reçue par le poste serveur. Lorsque ce mécanisme est activé, deux fichiers sont créés dans le dossier Logs de la base, à côté du fichier de structure. Il sont nommés 4DRequestsLog_N.txt et 4DRequestsLog_ProcessInfo_N.txt où N est le numéro séquentiel de l'historique. Une fois qu'un fichier atteint une taille de 10 Mo, il est refermé et un nouveau fichier est généré, avec un numéro séquentiel incrémenté. Si un fichier du même nom existe déjà, il est directement remplacé. Vous pouvez définir le numéro de départ de la séquence à l'aide du paramètre *valeur*. Ces fichiers texte stockent dans un format tabulé simple diverses informations concernant chaque requête : heure, numéro de process, utilisateur, taille de la requête, durée de traitement, etc. Ces informations sont particulièrement utiles en phase de mise au point de l'application ou à des fins statistiques. Elles peuvent être importées par exemple dans un logiciel tableur afin d'être traitées.
Enreg requêtes client	Entier long	45	**Portée** : Poste 4D distant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 ou de 1 à N (0 = ne pas enregistrer, 1 à N = numéro séquentiel, accolé au nom du fichier). **Description** : Démarrage ou arrêt de l'enregistrement des requêtes standard effectuées par le poste client 4D ayant exécuté la commande (hors requêtes Web). Par défaut, la valeur est 0 (pas d'enregistrement des requêtes). 4D vous permet d'enregistrer l'historique des requêtes effectuées par le poste client. Lorsque ce mécanisme est activé, deux fichiers sont créés sur le poste client, dans le sous-dossier Logs du dossier local de la base. Il sont nommés 4DRequestsLog_N.txt et 4DRequestsLog_ProcessInfo_N.txt où N est le numéro séquentiel de l'historique. Une fois que le fichier 4DRequestsLog atteint une taille de 10 Mo, il est refermé et un nouveau fichier est généré, avec un numéro séquentiel incrémenté. Si un fichier du même nom existe déjà, il est directement remplacé. Vous pouvez définir le numéro de départ de la séquence à l'aide du paramètre valeur. Ces fichiers texte stockent dans un format tabulé simple diverses informations concernant chaque requête : heure, numéro de process, taille de la requête, durée de traitement, etc. Ces informations sont particulièrement utiles en phase de mise au point de l'application ou à des fins statistiques.
Inversion des objets	Entier long	37	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0, 1 ou 2 (0 = mode désactivé, 1 = mode automatique, 2 = mode activé). **Description** : Configuration du mode "inversion des objets". Ce mode permet d'inverser en mode Application les formulaires, objets, menus, etc. lorsque la base est affichée sous Windows dans une langue de droite à gauche. Ce mode peut également être configuré dans la page Interface/langue droite à gauche des Propriétés de la base. • La valeur 0 indique que le mode n'est jamais activé, quelle que soit la configuration du système (correspond à la valeur Jamais dans les Propriétés de la base). • La valeur 1 indique que le mode est activé ou non en fonction de la configuration du système (correspond à la valeur Automatique dans les Propriétés de la base). • La valeur 2 indique que le mode est activé, quelle que soit la configuration du système (correspond à la valeur Toujours dans les Propriétés de la base). Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.
Jeu de caractères	Entier long	17	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Jointures chercher par formule	Entier long	49	**Portée** : Process courant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (utiliser paramétrages de la base), 1 (toujours utiliser les liens auto) ou 2 (utiliser les jointures SQL si possible). **Description** : Mode de fonctionnement des commandes **CHERCHER PAR FORMULE** et **CHERCHER PAR FORMULE DANS SELECTION** relatif à l'utilisation de "jointures SQL". Dans les bases de données créées à compter de la version 11.2 de 4D v11 SQL, ces commandes effectuent des jointures sur le modèle des jointures SQL. Ce mécanisme permet de modifier la sélection d'une table en fonction d'une recherche effectuée sur une autre table sans que les tables soient reliées par un lien automatique (condition nécessaire dans les versions précédentes de 4D). Le sélecteur Jointures chercher par formule vous permet de définir le mode de fonctionnement des commandes de recherche par formule pour le process courant : • 0 : utiliser les paramètres courants de la base (valeur par défaut). Dans les bases de données créées à compter de la version 11.2 de 4D v11 SQL, les "jointures SQL" sont toujours activées pour les recherches par formule. Dans les bases de données converties, ce mécanisme est inactivé par défaut pour des raisons des compatibilité mais peut être mis en oeuvre via une préférence. • 1 : toujours utiliser les liens auto (= fonctionnement des versions précédentes de 4D). Dans ce mode, un lien est nécessaire pour définir la sélection d'une table en fonction de recherches effectuées dans une autre table. 4D n'effectue pas de "jointures SQL". • 2 : utiliser les jointures SQL si possible (= fonctionnement par défaut des bases créées en version 11.2 et suivantes de 4D v11 SQL). Dans ce mode, 4D établit des "jointures SQL" pour les recherches par formule lorsque la formule s'y prête (à deux exceptions près, voir la description de la commande commandes **CHERCHER PAR FORMULE** ou **CHERCHER PAR FORMULE DANS SELECTION**.

Constante	Type	Valeur	Commentaire
			Note : Avec 4D en mode distant, les "jointures SQL" ne peuvent être utilisées que si les formules sont exécutées sur le serveur (elles doivent avoir accès aux enregistrements). Pour configurer le lieu d'exécution des formules, reportez-vous aux sélecteurs 46 et 47.
JSON fuseau horaire local	Entier long	85	**Portée** : Process courant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 = ignorer le fuseau horaire local, 1 (défaut) = tenir compte du fuseau horaire. **Description** : Par défaut, la conversion des dates 4D vers le format JSON tient compte du fuseau horaire local. Par exemple, la conversion de la date !23/08/2013! donne "2013-08-22T22:00:00Z" au format JSON lorsque l'opération est effectuée en France en été (GMT+2). Ce principe est conforme au fonctionnement standard de JavaScript. Ce fonctionnement peut être source d'erreurs si vous souhaitez envoyer des valeurs de date en JSON à une personne qui se trouve dans un autre fuseau horaire. C'est le cas par exemple pour l'exportation d'une table avec **Selection vers JSON** en France destiné à être réimporté aux USA avec **JSON VERS SELECTION**. Par défaut, les dates étant réinterprétées dans chaque fuseau horaire, les valeurs stockées dans la base seront différentes. Dans ce cas, vous pouvez modifier le mode de conversion des dates afin qu'il ne tienne pas compte du fuseau horaire en passant 0 dans ce sélecteur. La conversion de la date !23/08/2013! donnera alors "2013-08-23T00:00:00Z" dans tous les cas.
Limitation nombre journaux	Entier long	90	**Portée** : 4D local, 4D Server. **Conservé entre deux sessions** : Non **Valeurs possibles** : Toute valeur entière, 0 = conserver tous les journaux **Description** : Nombre maximum de fichiers à conserver par roulement pour chaque type de journal. Par défaut, tous les fichiers sont conservés. Si vous passez une valeur N, seuls les N fichiers les plus récents seront conservés, le plus ancien étant automatiquement effacé à la création d'un nouveau. Ce paramétrage s'applique à chacun des fichiers journaux suivants : journal des requêtes (sélecteurs 28 et 45), journal de débogage (sélecteur 34), journal des événements (sélecteur 79), ainsi que l'historique des requêtes Web et l'historique debug des requêtes Web (sélecteurs 29 et 84 de la commande **WEB FIXER OPTION**).
Liste commandes enreg	Chaîne	80	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Chaîne contenant la liste des numéros des commandes 4D à enregistrer (séparées par des points-virgules), "all" pour enregistrer toutes les commandes ou "" (chaîne vide) pour n'enregistrer aucune commande. **Description** : Liste des commandes 4D à enregistrer dans le fichier de débogage (cf. sélecteur 34, Enreg événements debogage). Par défaut, toutes les commandes 4D sont enregistrées. Ce sélecteur vous permet de restreindre la quantité d'informations stockées dans le fichier de débogage en limitant les commandes 4D dont vous souhaitez enregistrer l'exécution. Par exemple, vous pouvez écrire : `FIXER PARAMETRE BASE(Liste commandes enreg;"277;341") //enregistrer uniquement les commandes CHERCHER et CHERCHER DANS SELECTION`
Liste de chiffrement SSL	Chaîne	64	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Suite de chaînes séparées par des deux-points **Description :** Liste de chiffrement (*cipher list*) utilisée par 4D pour le protocole sécurisé. Cette liste permet de modifier la priorité des algorithmes de chiffrement mis en oeuvre par 4D. Par exemple, vous pouvez passer la chaîne suivante dans le paramètre *valeur* : "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". Pour une description complète de la syntaxe de la liste de chiffrement, reportez-vous à la page ciphers sur le site de OpenSSL. Ce paramétrage est global à l'application 4D (il concerne le serveur HTTP, le serveur SQL, les connexions client/serveur ainsi que le client HTTP et les commandes 4D faisant appel au protocole sécurisé). Lorsque la liste de chiffrement a été modifiée, vous devez redémarrer le serveur concerné pour que le nouveau paramétrage soit pris en compte. Pour réinitialiser la liste de chiffrement à sa valeur par défaut (stockée en dur dans le fichier SLI), appelez la commande **FIXER PARAMETRE BASE** et passez une chaîne vide ("") dans le paramètre *valeur*. **Note :** Avec la commande **Lire parametre base**, la liste de chiffrement est retournée dans le paramètre optionnel *valeurAlpha* et le paramètre de retour vaut toujours 0.
Maximum process Web	Entier long	7	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 -> 32 767 **Description** : Nombre maximum de process Web à maintenir en mode sans contexte avec 4D en mode local et 4D Server. Par défaut, la valeur est 10. Afin que le serveur Web soit réactif, en mode sans contexte, 4D maintient endormis les process Web pendant 5 secondes, et les réutilise pour traiter les éventuelles requêtes HTTP suivantes. En termes de performances, ce principe est en effet plus avantageux que la création d'un nouveau process à chaque requête. Une fois un process Web réutilisé, il est à nouveau endormi pour 5 secondes, sauf si le nombre maximum de process Web est dépassé (auquel cas il est tué). Si aucune requête n'a été attribuée à un process Web durant les 5 secondes, il est tué, sauf si un nombre minimum de process Web a été fixé et est atteint (auquel cas il est à nouveau endormi). Ces paramètres vous permettent d'ajuster le fonctionnement de votre serveur Web en fonction du nombre de requêtes, de la mémoire disponible, etc.
Minimum process Web	Entier long	6	**Portée** : 4D local, 4D Server **Conservé entre deux sessions :** Oui **Valeurs possibles** : 0 -> 32 767 **Description** : Nombre minimum de process Web à maintenir en mode sans contexte avec 4D en mode local et 4D Server. Par défaut, la valeur est 0 (cf. ci-dessous).
Mode Unicode	Entier long	41	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (mode compatibilité) ou 1 (mode Unicode) **Description** : Mode d'exécution courant de la base, relatif au jeu de caractères. 4D prend en charge le jeu de caractères Unicode mais peut fonctionner en mode "compatibilité" (basé sur le jeu de caractères ASCII Mac). Par défaut, les bases de données converties sont exécutées en mode compatibilité (0) et les bases créées à partir de la version 11 sont exécutées en mode Unicode. Le mode d'exécution est contrôlé via une option des préférences et peut également être lu ou (à des fins de test) modifié via ce sélecteur. La modification de cette option nécessite le redémarrage de la base pour être prise en compte. A noter que, dans le cadre d'un composant, il n'est pas possible de modifier cette valeur, mais uniquement de la lire.

Constante	Type	Valeur	Commentaire
Niveau de compression HTTP	Entier long	50	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Nombre de formules en cache	Entier long	92	**Portée** : Application 4D. **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif **Valeur par défaut** : 0 (pas de cache) **Description** : Fixe ou lit le nombre maximum de formules à conserver dans le cache des formules, qui est utilisé par la commande **EXECUTER FORMULE**. Cette limite est appliquée à tous les process, mais chaque process dispose de son propre cache de formules. Placer des formules dans le cache accélère l'exécution de la commande **EXECUTER FORMULE** en mode compilé puisque chaque formule en cache est tokenisée une seule fois dans ce cas. Lorsque vous modifiez la valeur du cache, son contenu est réinitialisé même si la nouvelle valeur est supérieure à la précédente. Une fois le nombre maximum de formules en cache atteint, toute nouvelle formule exécutée écrase la plus ancienne dans le cache (mode FIFO). Ce paramètre est pris en compte uniquement dans les bases ou les composants compilés.
Numéro automatique table	Entier long	31	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Valeurs possibles** : Toute valeur de type entier long. **Description** : Ce sélecteur permet de modifier ou de lire le numéro unique courant des enregistrements de la table passée en paramètre. “Numéro courant” signifie “dernier numéro utilisé” : si vous modifiez cette valeur à l’aide de **FIXER PARAMETRE BASE**, le prochain enregistrement sera créé avec comme numéro la valeur passée + 1. Ce nouveau numéro est, lui, retourné par la commande **Numerotation automatique** ainsi que dans tout champ de la table auquel la propriété "Incrémentation auto" a été affectée en Structure ou via le SQL. Par défaut, le numéro unique est défini par 4D et correspond à l’ordre de création des enregistrements. Pour des informations supplémentaires, reportez-vous à la documentation de la commande **Numerotation automatique**.
Numéro de port HTTPS	Entier long	39	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Numéro de port serveur SQL	Entier long	88	**Portée** : 4D mode local et 4D Server. **Conservé entre deux sessions** : Oui **Description** : Permet de lire ou de fixer le numéro du port TCP utilisé par le serveur SQL intégré de 4D en mode local ou de 4D Server. Par défaut, la valeur est 19812. Le numéro de port TCP peut également être défini dans la page "SQL" de la boîte de dialogue des Propriétés de la base. Lorsque ce sélecteur est utilisé en écriture, la propriété de la base est mise à jour. **Valeurs possibles** : 0 à 65535. **Valeur par défaut** : 19812
Numéro du port	Entier long	15	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Non **Description** : Numéro du port TCP utilisé par le serveur Web 4D avec 4D en mode local et 4D Server. Par défaut, la valeur est 80. Le numéro de port TCP est défini dans la page “Web/Configuration” de la boîte de dialogue des Propriétés de la base. Vous pouvez utiliser les constantes du thème **Numéros de port TCP** pour le paramètre *valeur*. Le sélecteur Numéro du port est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d’accès au mode Développement). Pour plus d’informations sur le numéro de port TCP, reportez-vous à la section **Paramétrages du serveur Web**.
Numéro du port client serveur	Entier long	35	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 à 65535 **Description** : Numéro de port TCP sur lequel 4D Server publie la base de données (à destination des postes 4D distants). Par défaut, la valeur est 19813. La personnalisation de cette valeur permet d’utiliser plusieurs applications 4D client-serveur sur la même machine avec le protocole TCP ; dans ce cas, vous devez spécifier un numéro de port différent pour chaque application. La valeur est stockée dans le fichier de structure de la base. Elle peut être définie avec 4D en mode local mais n’est prise en compte qu’en configuration client-serveur. Lorsque vous modifiez cette valeur, il est nécessaire de redémarrer le poste serveur afin que la nouvelle valeur soit prise en compte.
Périodicité écriture cache	Entier long	95	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Non **Valeurs possibles** : entier long > 1 (secondes) **Description** : Permet de lire ou de fixer la valeur courante de périodicité de l'écriture du cache de données sur le disque, exprimée en secondes. Si elle est modifiée, cette valeur remplace la valeur définie par l'option **Ecriture cache toutes les <n> secondes/minutes** dans la **Page Base de données/Mémoire** des Propriétés de la base durant la session courante (elle n'est pas stockée dans les Propriétés de la base).
PHP adresse IP interpréteur	Entier long	55	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Chaîne formatée du type "nnn.nnn.nnn.nnn" (par exemple "127.0.0.1"). **Description** : Adresse IP utilisée localement par 4D pour communiquer avec l’interpéteur PHP via fastcgi. Par défaut, la valeur est "127.0.0.1". Cette adresse doit correspondre à la machine sur laquelle se trouve 4D. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d’informations sur l’interpréteur PHP de 4D, reportez-vous au manuel *Mode Développement*.
PHP nombre	Entier		**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Valeur de type entier long positif. Par défaut, la valeur est 5. **Description** : Nombre de process enfants à créer et à maintenir localement par l’interpréteur PHP de 4D. Pour des raisons d’optimisation, l’interpréteur PHP crée et utilise un ensemble (pool) de process système appelés "process enfants" pour traiter les demandes d’exécution de scripts. Vous pouvez faire varier le nombre de process enfants en

Constante	Type	Valeur	Commentaire
Défaut	Type	57	fonction des besoins de votre application. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations sur l'interpréteur PHP de 4D, reportez-vous au manuel *Mode Développement*. **Note :** Sous Mac OS, tous les process enfants partagent le même port. Sous Windows, chaque process enfant utilise un numéro de port spécifique. Le premier numéro est celui défini pour l'interpréteur PHP, les autres process enfants l'incrémentent. Par exemple, si le port par défaut est le 8002 et que vous lancez 5 process enfants, ils utiliseront les ports 8002 à 8006.
PHP nombre requêtes max	Entier long	58	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Valeur de type entier long positif. Par défaut, la valeur est 500. **Description** : Nombre maximum de requêtes acceptées par l'interpréteur PHP. Lorsque ce nombre maximum est atteint, l'interpréteur retourne des erreurs du type "serveur occupé". Pour des raisons de sécurité ou de performance, vous pouvez modifier cette valeur. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations, reportez-vous à la documentation de fastcgi-php. **Note :** Côté 4D, ces paramètres sont appliqués dynamiquement, il n'est pas nécessaire de quitter le programme pour leur prise en compte. En revanche, si l'interpréteur PHP était déjà lancé, il est nécessaire de le quitter et de le relancer pour qu'il prenne en compte ces modifications.
PHP port interpréteur	Entier long	56	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : Valeur de type entier long positif. Par défaut, la valeur est 8002. **Description** : Numéro du port TCP utilisé par l'interpréteur PHP de 4D. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base. Pour plus d'informations sur l'interpréteur PHP de 4D, reportez-vous au manuel *Mode Développement*.
PHP utiliser interpréteur externe	Entier long	60	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs** : 0 = utiliser interpréteur interne, 1 = utiliser interpréteur externe **Description** : Valeur indiquant si les requêtes PHP de 4D sont adressées à l'interpréteur interne fourni par 4D ou un interpréteur externe. Par défaut, la valeur est 0 (utilisation de l'interpréteur fourni par 4D). Si vous souhaitez utiliser votre propre interpréteur PHP, par exemple pour bénéficier de modules supplémentaires ou d'une configuration spécifique, passez 1 dans *valeur*. Dans ce cas, 4D ne démarrera pas son interpréteur en cas de requête PHP. L'interpréteur PHP personnalisé doit avoir été compilé en fastcgi et se trouver sur la même machine que le moteur 4D. A noter que dans ce cas, vous devez entièrement gérer l'interpréteur, il n'est ni démarré ni stoppé par 4D. Ce paramètre peut également être défini globalement pour tous les postes via les Propriétés de la base.
Prise en charge QuickTime	Entier long	82	**Portée** : Application 4D **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (défaut) = QuickTime désactivé, 1 = QuickTime activé. **Description** : Dans 4D à compter de la v14, par défaut les codecs QuickTime ne sont plus pris en charge. Par compatibilité, vous pouvez les réactiver dans votre base à l'aide de ce sélecteur. La modification de cette option nécessite le redémarrage de la base. A noter toutefois que la prise en charge de QuickTime sera définitivement supprimée dans les prochaines versions de 4D.
Process Web simultanés maxi	Entier long	18	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Seuil de compression HTTP	Entier long	51	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
SQL autocommit	Entier long	43	**Portée** : Base de données **Conservé entre deux sessions** : Oui **Valeurs possibles** : 0 (désactivation) ou 1 (activation) **Description** : Activation ou désactivation du mode SQL auto-commit. Par défaut, la valeur est 0 (mode désactivé) Le mode auto-commit permet de renforcer l'intégrité référentielle de la base. Lorsque ce mode est actif, les requêtes **SELECT**, **INSERT**, **UPDATE**, **DELETE** (SIUD) sont automatiquement incluses dans des transactions lorsqu'elles sont exécutées en-dehors de toute transaction. Ce mode peut également être défini dans les préférences de la base.
Synchro auto dossier Resources	Entier long	48	**Portée** : Poste 4D distant **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (pas de synchronisation), 1 (synchronisation auto) ou 2 (demander). **Description** : Mode de synchronisation dynamique du dossier *Resources* du poste client 4D ayant exécuté la commande avec celui du serveur. Lorsque le contenu du dossier *Resources* sur le serveur a été modifié ou qu'une demande de synchronisation a été émise (via l'explorateur de ressources ou suite à l'exécution de la commande **NOTIFIER MODIFICATION DOSSIER RESOURCES**), le serveur notifie les clients connectés. Trois modes de synchronisation sont alors possibles côté client. Le sélecteur Synchro auto dossier Resources vous permet de définir le mode à utiliser pour le poste client et la session courante : • 0 (valeur par défaut) : pas de synchronisation dynamique (la demande de synchronisation est ignorée) • 1 : synchronisation dynamique automatique • 2 : affichage d'une boîte de dialogue sur les postes clients, avec possibilité d'effectuer ou de refuser la synchronisation. Le mode de synchronisation peut également être défini globalement dans les Propriétés de la base.
Taille maxi mémoire	Entier	61	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif. **Description** : Taille maximum de mémoire temporaire que 4D pourra allouer à chaque process, exprimée en Mo. Par défaut, la valeur est 0 (pas de taille maximum). 4D utilise une mémoire temporaire spéciale dédiée aux opérations d'indexation et de tri. Cette mémoire a pour but de préserver la mémoire cache "classique" lors d'opérations massives. Elle n'est activée qu'en cas de besoin. Par défaut, la taille de la mémoire temporaire n'est limitée que par les ressources disponibles (en fonction de la configuration mémoire du système).

Constante	Type	Valeur	Commentaire
Taille maximum requêtes Web	Entier long	27	Ce mécanisme convient à la plupart des applications. Toutefois, dans certains contextes spécifiques, notamment lorsqu'une application client-serveur effectue simultanément un grand nombre de tris séquentiels, la taille de la mémoire temporaire peut augmenter de façon critique, jusqu'à rendre le système instable. Dans ce contexte, fixer une taille maximum à la mémoire temporaire permet de préserver le fonctionnement de l'application. En contrepartie, la vitesse d'exécution pourra être affectée : lorsque la taille maximum est atteinte pour un process, 4D utilise des fichiers disque, ce qui peut ralentir les traitements. Pour des besoins tels que ceux décrits ci-dessus, une taille maximum d'environ 50 Mo est généralement un bon compromis. La valeur idéale sera cependant à déterminer en fonction des spécificités de l'application et résultera généralement de tests en volumétrie réelle. **Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : *Constante obsolète (conservée par compatibilité uniquement).* Il est désormais conseillé d'utiliser les commandes **WEB FIXER OPTION** et **WEB LIRE OPTION** pour le paramétrage du serveur HTTP.
Taille minimum libération cache	Entier long	66	**Portée** : Application 4D **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif > 1. **Description** : Taille minimum de mémoire à libérer du cache de la base de données lorsque le moteur a besoin d'y faire de la place pour y allouer un objet (valeur en octets). Ce sélecteur a pour but de permettre de réduire le nombre de libérations de données du cache afin d'obtenir des gains de performances. Vous pouvez faire varier ce paramétrage en fonction de la taille du cache et de celle des blocs de données manipulées dans votre base. Par défaut, si ce sélecteur n'est pas utilisé, 4D décharge au minimum 10 % du cache en cas de besoin de place.
Taille pile process base server	Entier long	53	**Portée** : 4D Server **Conservé entre deux sessions** : Non **Valeurs possibles** : Entier long positif. **Description** : Taille de la pile allouée à chaque process système préemptif sur le serveur, exprimée en octets. La taille par défaut est déterminée par le système. Les process système préemptifs (process de type Process base 4D client) sont chargés de contrôler les process clients 4D principaux. La taille allouée par défaut à la pile de chaque process préemptif permet un bon confort d'exécution mais peut s'avérer conséquente lorsque de très nombreux process (plusieurs centaines) sont créés. A des fins d'optimisation, cette taille peut être diminuée sensiblement si les opérations effectuées par la base s'y prêtent (par exemple si la base n'effectue pas de tris sur de grosses quantités d'enregistrements). Des valeurs de 512 voire de 256 Ko sont possibles. Attention, le sous-dimensionnement de la pile est critique et peut nuire au fonctionnement de 4D Server. Le réglage de ce paramètre est à effectuer avec précaution et doit tenir compte des conditions d'utilisation de la base (nombre d'enregistrements, types d'opérations, etc.). Pour être pris en compte, ce paramétrage doit être exécuté sur le poste serveur (par exemple dans la méthode base Sur démarrage serveur).
Timeout 4D mode distant	Entier long	14	**Portée** (ancienne couche réseau uniquement) : Application 4D si *valeur* positive **Conservé entre deux sessions** : Oui si *valeur* positive **Description** : A utiliser dans des cas très spécifiques. Valeur du délai avant déconnexion (timeout) accordé par le poste 4D distant au poste 4D Server. Par défaut, cette valeur est définie dans la page "Client-Serveur/Options réseau" des Propriétés de la base, sur le poste distant. Le sélecteur Timeout 4D mode distant n'est pris en compte que si vous utilisez l'ancienne couche réseau. Avec la couche *ServerNet* activée, il est ignoré : ce paramétrage est entièrement géré par le sélecteur Timeout 4D Server (13).
Timeout 4D Server	Entier long	13	**Portée** : Application 4D si *valeur* positive **Conservé entre deux sessions** : Oui si *valeur* positive **Valeurs possibles** : 0 -> 32 767 **Description** : Valeur du délai avant déconnexion (timeout) accordé par 4D Server aux postes clients. Par défaut, cette valeur est définie dans la page "Client-Serveur/Options réseau" des Propriétés de la base, sur le poste serveur. Le timeout serveur définit la période maximale de non-réponse du client "autorisée", par exemple s'il effectue une opération bloquante. A l'issue de cette période, 4D Server déconnecte le client. Le sélecteur Timeout 4D Server vous permet de fixer un nouveau timeout, exprimé en minutes. Cette possibilité permet en particulier d'augmenter la valeur du timeout avant l'exécution sur le poste client d'une opération bloquante de longue durée, risquant d'entraîner une déconnexion ; par exemple, l'impression d'un grand nombre de pages. Vous disposez en outre de deux possibilités : • effectuer une modification globale et permanente : la nouvelle valeur s'applique à tous les process et est stockée dans les préférences de l'application (équivaut à une modification de la valeur dans la boîte de dialogue des Préférences). Pour cela, passez une valeur **positive** dans le paramètre *valeur*. • effectuer une modification restreinte et temporaire : la nouvelle valeur ne s'applique qu'au process appelant (les autres process conservant la valeur d'origine), et est abandonnée dès que le serveur reçoit un signe d'activité du poste client — par exemple, dès que l'opération est terminée. Cette possibilité est utile pour gérer les opérations longues initiées par des plug-ins. Pour cela, passez une valeur **négative** dans le paramètre *valeur*. Pour définir une connexion "Ouverte en permanence", passez 0 dans *valeur*. Reportez-vous à l'exemple 1.
Timeout connexions inactives	Entier long	54	**Portée** : Application 4D sauf si valeur négative **Conservé entre deux sessions** : Non **Valeurs possibles** : Valeur entière exprimant une durée en secondes. La valeur peut être positive (nouvelles connexions) ou négative (connexions existantes). Par défaut, la valeur est 20. **Description** : Délai maximum d'inactivité (timeout) des connexions au moteur de base de données et au serveur SQL de 4D ainsi que, en mode *ServerNet* (nouvelle couche réseau), au serveur d'applications 4D. Lorsqu'une connexion inactive atteint ce délai, elle est automatiquement mise en veille, ce qui se traduit par le gel de la session client/serveur et la fermeture du socket réseau. Dans la fenêtre d'administration du serveur, le process utilisateur prend l'état "Postponed". Ce fonctionnement est entièrement transparent pour l'utilisateur : dès qu'il y a reprise d'activité sur la connexion mise en veille, le socket est automatiquement rouvert et la session client/serveur restaurée. Ce paramétrage permet, d'une part, d'économiser des ressources sur le serveur : les connexions mises en veille referment le socket et libèrent un processus sur le serveur. D'autre part, il permet d'éviter les pertes de connexions dues aux fermetures par les pare-feux des sockets inactifs. La valeur de timeout des connexions inactives doit pour cela être inférieure à celle du pare-feu. Si vous passez une valeur positive dans *valeur*, elle s'applique à toutes les nouvelles connexions dans tous les process. Si vous passez une valeur négative, elle s'applique aux connexions ouvertes dans le process courant. Si vous passez 0, les connexions inactives ne sont pas soumises à un timeout. Ce paramètre peut être défini côté serveur et côté client. Si vous passez deux durées différentes, la plus courte sera

Constante	Type	Valeur	Commentaire
			prise en compte. Généralement, vous n'aurez pas besoin de modifier cette valeur.
Trier par formule serveur	Entier long	47	**Portée** : Table et process courants **Conservé entre deux sessions** : Non **Valeurs possibles** : 0 (utiliser le paramétrage de la base), 1 (exécuter sur le client) ou 2 (exécuter sur le serveur) **Description** : Emplacement de l'exécution de la commande **TRIER PAR FORMULE** pour la table passée en paramètre. Dans le cadre de l'exploitation d'une base en client-serveur, la commande **TRIER PAR FORMULE** peut être exécutée soit sur le serveur soit sur le client. Ce sélecteur permet de définir l'emplacement de l'exécution de cette commande (serveur ou client). Ce mode peut également être défini dans les préférences de la base. Pour plus d'informations, reportez-vous à la description du sélecteur 46, Chercher par formule serveur. **Note :** Si vous souhaitez pouvoir activer les jointures "type SQL" (cf. sélecteur Jointures chercher par formule), vous devez toujours exécuter les formules sur le serveur afin qu'elle ait accès aux enregistrements. Attention, dans ce contexte, la formule ne doit pas contenir d'appel à une méthode, sinon elle est automatiquement basculée sur le poste distant.
Utiliser ancienne couche réseau	Entier long	87	**Portée :** 4D local, 4D Server. **Conservé entre deux sessions :** Oui **Description :** Fixe ou lit le statut courant de l'ancienne couche réseau pour les connexions client/serveur. L'ancienne couche réseau est obsolète à compter de 4D v14 R5 et doit être progressivement remplacée dans vos applications par la couche réseau *ServerNet*. *ServerNet* sera nécessaire dans les prochaines versions de 4D afin de permettre aux applications 4D de tirer parti des futures évolutions réseau. Pour des raisons de compatibilité, l'ancienne couche réseau est toujours prise en charge afin de faciliter la transition des applications existantes (elle reste utilisée par défaut dans les applications converties depuis des versions antérieures à la v14 R5). Passez 1 dans ce paramètre pour utiliser l'ancienne couche réseau (et désactiver *ServerNet*), et passez 0 pour désactiver l'ancienne couche réseau (et utiliser *ServerNet*). Cette propriété peut également être définie à l'aide de l'option "Utiliser l'ancienne couche réseau" présente dans la **Page Compatibilité** des Propriétés de la base (voir section **Options réseau et Client-serveur** ; dans cette section, vous trouverez aussi un paragraphe décrivant la stratégie de migration. Nous vous recommandons d'activer *ServerNet* dès que possible). Il est nécessaire de redémarrer l'application pour que ce paramètre soit pris en compte. Cette option n'est pas disponible dans 4D Server v14 R5 version 64 bits pour OS X, qui prend en charge uniquement *ServetNet* (elle vaut toujours 0). **Valeurs possibles :** 0 ou 1 (0 = ne pas utiliser l'ancienne couche, 1 = utiliser l'ancienne couche) **Valeur par défaut :** 0 dans les applications créées avec 4D v14 R5 ou suivantes, 1 dans les applications converties depuis 4D v14 R4 ou précédentes.

PHP

Constante	Type	Valeur	Comment
PHP privilèges	Entier long	1	Définition des privilèges utilisateur spécifiques relatifs à l'exécution du script. **Valeur(s) possible(s)** : Chaîne de la forme "Utilisateur:Mot de passe". Par exemple : "root:jd51254d"
PHP résultat brut	Entier long	2	Définition du mode de traitement des en-têtes HTTP renvoyés par PHP dans le résultat de l'exécution lorsque ce résultat est de type texte (lorsque le résultat est de type BLOB, les en-têtes sont toujours conservés). **Valeur(s) possible(s)** : Booléen. Faux (valeur par défaut) = supprimer les en-têtes HTTP du résultat. Vrai = conserver les en-têtes HTTP.

PROFONDEUR ECRAN

Constante	Type	Valeur	Comment
Deux cent cinquante six coul	Entier long	8	
Est en couleurs	Entier long	1	
Est en niveaux de gris	Entier long	0	
Milliers de couleurs	Entier long	16	
Millions de couleurs 24 bits	Entier long	24	
Millions de couleurs 32 bits	Entier long	32	
Noir et blanc	Entier long	0	
Quatre couleurs	Entier long	2	
Seize couleurs	Entier long	4	

Propriétés des lignes de menu

Constante	Type	Valeur	Comment
Action standard associée	Chaîne	4D_standard_action	Associer une action standard à la ligne de menu Voir les constantes du thème "**Valeurs pour Actions standard associée**"
Autorisations d'accès	Chaîne	4D_access_group	Affecter un groupe d'accès à la commande 0 = Sans restriction >0 = Numéro de groupe
Démarrer un process	Chaîne	4D_start_new_process	Activer l'option "Démarrer un nouveau process" 0 = Non, 1 = Oui

Propriétés des ressources

Constante	Type	Valeur	Comment
Bit ressource heap système	Entier long	6	
Bit ressource modifiée	Entier long	1	
Bit ressource préchargée	Entier long	2	
Bit ressource protégée	Entier long	3	
Bit ressource purgeable	Entier long	5	
Bit ressource verrouillée	Entier long	4	
Masque ressource heap système	Entier long	64	
Masque ressource modifiée	Entier long	2	
Masque ressource préchargée	Entier long	4	
Masque ressource protégée	Entier long	8	
Masque ressource purgeable	Entier long	32	
Masque ressource verrouillée	Entier long	16	

Propriétés plate-forme

Les constantes préfixées _o_ sont obsolètes et ne peuvent plus être retournées par la commande. Elles sont conservées par compatibilité.

Constante	Type	Valeur	Comment
_o_INTEL 386	Entier long	386	
_o_INTEL 486	Entier long	486	
_o_Macintosh 68K	Entier long	1	
_o_PowerPC 601	Entier long	601	
_o_PowerPC 603	Entier long	603	
_o_PowerPC 604	Entier long	604	
_o_PowerPC G3	Entier long	510	
Compatible Intel	Entier long	586	
Mac OS	Entier long	2	
Power PC	Entier long	406	
Windows	Entier long	3	

QR Commandes

Constante	Type	Valeur	Comment
qr cmd ajouter colonne	Entier long	2608	
qr cmd aligné à droite	Entier long	505	
qr cmd aligné à gauche	Entier long	503	
qr cmd aligné par défaut	Entier long	512	
qr cmd apercu	Entier long	2007	
qr cmd cacher colonne	Entier long	2602	
qr cmd cacher ligne	Entier long	2607	
qr cmd centré	Entier long	504	
qr cmd déplacer à droite	Entier long	3003	
qr cmd déplacer à gauche	Entier long	3002	
qr cmd destination 4D View	Entier long	2503	
qr cmd destination fichier	Entier long	2501	
qr cmd destination fichier HTML	Entier long	2504	
qr cmd destination graphe	Entier long	2502	
qr cmd destination imprimante	Entier long	2500	
qr cmd écart type	Entier long	513	
qr cmd éditer colonne	Entier long	2603	
qr cmd encadrements	Entier long	2609	
qr cmd enregistrer	Entier long	2002	
qr cmd enregistrer sous	Entier long	2003	
qr cmd entete et pied de page	Entier long	2005	
qr cmd espacement des totaux	Entier long	2610	
qr cmd executer	Entier long	2008	
qr cmd format	Entier long	2606	
qr cmd gras	Entier long	500	
qr cmd insérer colonne	Entier long	2600	
qr cmd italique	Entier long	501	
qr cmd largeur automatique	Entier long	2605	
qr cmd liste déroulante police	Entier long	1000	
qr cmd max	Entier long	509	
qr cmd min	Entier long	508	
qr cmd mise en forme	Entier long	2611	
qr cmd mise en page	Entier long	2006	
qr cmd moyenne	Entier long	507	
qr cmd nombre	Entier long	510	
qr cmd normal	Entier long	511	
qr cmd nouveau	Entier long	2000	
qr cmd objet couleur fond	Entier long	1003	
qr cmd objet couleur fond alt	Entier long	1004	
qr cmd objet couleur texte	Entier long	1002	
qr cmd ouvrir	Entier long	2001	
qr cmd palette colonnes	Entier long	2054	
qr cmd palette couleurs de fond	Entier long	2052	
qr cmd palette opérateurs	Entier long	2051	
qr cmd palette standard	Entier long	2053	
qr cmd palette style	Entier long	2050	
qr cmd somme	Entier long	506	
qr cmd souligné	Entier long	502	
qr cmd supprimer colonne	Entier long	2601	
qr cmd valeurs répétées	Entier long	2604	
qr cmd version enregistrée	Entier long	2004	

QR Destination de sortie

Constante	Type	Valeur	Comment
_o_qr zone 4D Chart	Entier long	4	*** Constante obsolète ***
qr fichier HTML	Entier long	5	*spécificités* : Chemin d'accès du fichier.
qr fichier texte	Entier long	2	*spécificités* : Chemin d'accès du fichier.
qr imprimante	Entier long	1	*spécificités* : "*" pour supprimer les boîtes de dialogue d'impression
qr zone 4D View	Entier long	3	*spécificités* : N.A.

QR Encadrements

Constante	Type	Valeur	Comment
qr encadrement bas	Entier long	8	Bordure inférieure
qr encadrement droit	Entier long	4	Bordure droite
qr encadrement gauche	Entier long	1	Bordure gauche
qr encadrement haut	Entier long	2	Bordure supérieure
qr encadrement horiz intérieur	Entier long	32	Bordure intérieure horizontale
qr encadrement vert intérieur	Entier long	16	Bordure intérieure verticale

QR Lignes pour Propriétés

Constante	Type	Valeur	Comment
qr détail	Entier long	-2	Zone Détail de l'état
qr entête	Entier long	-4	En-tête de page
qr intitulé	Entier long	-1	Intitulé de l'état
qr pied de page	Entier long	-5	Pied de page
qr total général	Entier long	-3	Zone Total général

QR Opérateurs

Constante	Type	Valeur	Comment
qr écart type	Entier long	32	
qr max	Entier long	8	
qr min	Entier long	4	
qr moyenne	Entier long	2	
qr nombre	Entier long	16	
qr somme	Entier long	1	

QR Propriétés de document

Constante	Type	Valeur	Comment
qr dialogue d'impression	Entier long	1	Affichage de la boîte de dialogue d'impression : • Si valeur = 0, la boîte de dialogue d'impression n'est pas affichée avant l'impression. • Si valeur = 1, la boîte de dialogue d'impression est affichée avant l'impression (valeur par défaut).
qr unité	Entier long	2	Unité du document : • Si valeur = 0, l'unité du document est le point. • Si valeur = 1, l'unité du document est le centimètre. • Si valeur = 2, l'unité du document est le pouce.

QR Propriétés de texte

Constante	Type	Valeur	Comment
_o_qr police	Entier long	1	Obsolète depuis 4D v14R3 (utiliser qr nom de police)
qr couleur de fond	Entier long	8	Numéro de couleur de fond
qr couleur de fond alternée	Entier long	9	Numéro de couleur de fond alternée
qr couleur de texte	Entier long	6	Numéro de couleur (Entier long)
qr gras	Entier long	3	Attribut gras (0 ou 1)
qr italique	Entier long	4	Attribut italique (0 ou 1)
qr justification	Entier long	7	Attribut de justification (0 = par défaut, 1 = gauche, 2 = centre et 3 = droite)
qr nom de police	Entier long	10	Nom de police tel que retourné par exemple par la commande **LISTE DES POLICES**.
qr souligné	Entier long	5	Attribut souligné (0 ou 1)
qr taille police	Entier long	2	Taille de police en points (9 à 255)

QR Propriétés de zone

Constante	Type	Valeur	Comment
qr barre menu	Entier long	1	Affichage de la barre de menus (Affichée=1, Cachée=0)
qr barre outils colonnes	Entier long	6	Affichage de la barre d'outils Colonnes (Affichée=1, Cachée=0)
qr barre outils couleurs	Entier long	5	Affichage de la barre d'outils Couleurs de fond (Affichée=1, Cachée=0)
qr barre outils opérateurs	Entier long	4	Affichage de la barre d'outils Opérateurs (Affichée=1, Cachée=0)
qr barre outils standard	Entier long	2	Affichage de la barre d'outils standard (Affichée=1, Cachée=0)
qr barre outils style	Entier long	3	Affichage de la barre d'outils Style (Affichée=1, Cachée=0)
qr menus contextuels	Entier long	7	Affichage des menus contextuels (Affichés=1, Cachés=0)

QR Types d'états

Constante	Type	Valeur	Comment
qr état en liste	Entier long	1	
qr état tableau croisé	Entier long	2	

Recherches

Constante	Type	Valeur	Comment
Description format Texte	Entier long	0	
Description format XML	Entier long	1	
Vers ensemble	Entier long	1	
Vers sélection courante	Entier long	0	
Vers sélection temporaire	Entier long	2	
Vers variable	Entier long	3	

Sauvegarde et restitution

Constante	Type	Valeur	Comment
Attribut champ nom	Entier long	2	Les champs sont identifiés par leur nom. Exemple: {"Nom":"Jones"}
Attribut champ numéro	Entier long	1	Les champs sont identifiés par leur numéro (défaut si omis). Exemple: {"5":"Jones"}.
Date dernière restitution	Entier long	0	
Date dernière sauvegarde	Entier long	0	
Date prochaine sauvegarde	Entier long	4	
Mode réparation auto	Entier long	1	Utiliser le mode flexible avec réparation automatique et remplir le paramètre *objErreur* (si passé)
Mode strict	Entier long	0	Utiliser le mode d'intégration avec contrôle strict des opérations (option par défaut). Recommandé dans la plupart des cas.
Statut dernière restitution	Entier long	2	
Statut dernière sauvegarde	Entier long	2	

Serveur Web

Constante	Type	Valeur	Comment
wdl activer avec body request	Entier long	5	Le fichier de debug de requêtes HTTP Web est activé avec le body de la requête uniquement
wdl activer avec body response	Entier long	3	Le fichier de debug de requêtes HTTP Web est activé avec le body de la réponse uniquement
wdl activer avec tous body	Entier long	7	Le fichier de debug de requêtes HTTP Web est activé avec les body de la requête et de la réponse
wdl activer sans body	Entier long	1	Le fichier de debug de requêtes HTTP Web est activé sans les body (la taille des body est fournie dans ce cas)
wdl désactiver log	Entier long	0	Le fichier de debug de requêtes HTTP Web est désactivé
Web activer validation adresse IP de session	Entier long	83	**Portée** : serveur Web local **Conservé entre deux sessions** : Non **Description** : Permet de désactiver ou d'activer la validation des adresses IP pour les cookies de session. Pour des raisons de sécurité, par défaut le serveur Web de 4D vérifie l'adresse IP de chaque requête contenant un cookie de session et rejette la requête si cette adresse ne correspond pas à l'adresse IP utilisée pour créer le cookie. Dans certaines applications spécifiques, vous pouvez souhaiter désactiver cette validation et accepter les cookies de session même si les adresses IP ne correspondent pas. Par exemple, lorsque les appareils mobiles basculent du réseau Wifi au réseau 3G/4G, leur adresse IP change. Dans ce cas, vous pouvez passer 0 dans cette option afin de permettre aux clients de pouvoir continuer à utiliser leur session Web même après changement d'adresse IP. Notez que ce paramétrage abaisse le niveau de sécurité de votre application. Lorsqu'il est modifié, ce paramétrage est effectif immédiatement (il n'est pas nécessaire de redémarrer le serveur HTTP). **Valeurs possibles** : 0 (désactivé) ou 1 (activé) **Valeur par défaut** : 1 (les adresses IP sont vérifiées)
Web adresse IP d'écoute	Entier long	16	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Adresse IP sur laquelle le serveur Web doit recevoir les requêtes HTTP avec 4D en mode local et 4D Server. Par défaut, aucune adresse particulière n'est spécifiée (*valeur* = 0). Ce paramètre est défini dans les Propriétés de la base. Le sélecteur *Web Adresse IP d'écoute* est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d'accès au mode Développement). Le paramètre *valeur* contient l'adresse IP sous forme hexadécimale. Ainsi, pour désigner une adresse du type "a.b.c.d", le code sera de la forme : `C_ENTIER LONG($addr)` `$addr:=($a<<24)\|($b<<16)|($c<<8)\|$d` `WEB FIXER OPTION(Web Adresse IP découte;$addr)`
Web chemin du cookie de session	Entier long	82	**Portée :** serveur Web local **Conservé entre deux sessions :** Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de fixer ou de lire la valeur du champ "path" du cookie de session. Ce sélecteur (ainsi que le sélecteur 81) est utile pour contrôler la portée des cookies de session : si vous définissez par exemple la valeur "/4DACTION" pour ce sélecteur, le client n'enverra un cookie que pour les requêtes dynamiques débutant par 4DACTION et pas pour les images, pages statiques, etc. **Valeurs :** Texte
Web conserver les sessions	Entier long	70	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet d'activer ou d'inactiver la gestion automatique des sessions (décrit dans la section **Gestion des sessions Web**). **Valeurs** : 1 (activer mode) ou 0 (inactiver mode) **Valeur par défaut** : 1 pour les bases créées depuis la v13, 0 pour les bases converties. A noter que ce mode active également le mécanisme de réutilisation des contextes temporaires en mode distant. Pour plus d'informations sur ce mécanisme, reportez-vous à la description de cette option dans la section **Paramétrages du serveur Web**.
Web debug log	Entier long	84	**Portée :** serveur Web local **Conservé entre deux sessions :** Non, mais reste valide même si le serveur HTTP est redémarré (un nouveau fichier est utilisé) **Description :** Permet de lire ou de définir le statut du fichier de debug de requêtes HTTP du serveur Web 4D. Lorsqu'il est activé, ce fichier, nommé "**HTTPDebugLog_*nn*.txt**", est stocké dans le dossier "Logs" de l'application (*nn* est le numéro de fichier). L'historique des requêtes HTTP est particulièrement utile dans le contexte du débogage du serveur Web. Il stocke en texte brut chaque requête et chaque réponse. La totalité des requêtes, en-têtes compris, est enregistrée ; en option, il est possible d'enregistrer également le corps (body) des requêtes. **Valeurs** : Une des constantes préfixées "wdl" (reportez-vous ci-dessous à la description de ces constantes). **Valeur par défaut** : 0 (non activé)
Web domaine du cookie de session	Entier long	81	**Portée :** serveur Web local **Conservé entre deux sessions :** Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de fixer ou de lire la valeur du champ "domain" du cookie de session. Ce sélecteur (ainsi que le sélecteur 82) est utile pour contrôler la portée des cookies de session : si vous définissez par exemple la valeur "/*.4d.fr" pour ce sélecteur, le client n'enverra un cookie que lorsque la requête s'adresse au domaine ".4d.fr", ce qui exclut les serveurs hébergeant des données statiques externes. **Valeurs :** Texte
			Portée : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Démarrage ou arrêt de l'enregistrement des requêtes Web reçues par le serveur Web de 4D en mode local ou de 4D Server. Par défaut, la valeur est 0 (pas d'enregistrement des requêtes).

Constante	Type	Valeur	Commentaire
Web enreg requêtes	Entier long	29	L'historique des requêtes Web est stocké sous la forme d'un fichier texte nommé "logweb.txt" automatiquement placé dans le dossier Logs de la base, à côté du fichier de structure. Le format de ce fichier est déterminé par la valeur que vous passez. Pour plus d'informations sur les différents formats d'historique des requêtes, reportez-vous à la section **Informations sur le site Web**. L'activation de ce fichier peut également être définie dans la page "Web/Journal" des Propriétés de la base. **Valeurs possibles** : 0 = Ne pas enregistrer (défaut), 1 = Enregistrer au format CLF, 2 = Enregistrer au format DLF, 3 = Enregistrer au format ELF, 4 = Enregistrer au format WLF. **Attention** : Les formats 3 et 4 sont des formats personnalisés, dont le contenu doit être défini au préalable dans les Propriétés de la base. Si vous utilisez l'un de ces formats sans que ses champs n'aient été sélectionnés, le fichier des requêtes n'est pas généré.
Web jeu de caractères	Entier long	17	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Jeu de caractères que le serveur Web 4D (avec 4D en mode local et 4D Server) utilise pour communiquer avec les navigateurs Web qui se connectent à la base. La valeur par défaut dépend de la langue du système d'exploitation. Ce paramètre est défini dans les Propriétés de la base. **Valeurs** : Les valeurs possibles dépendent du mode d'exécution de la base relatif au jeu de caractères. • *Mode Unicode* : Lorsque l'application est exécutée en mode Unicode, les valeurs à passer pour ce paramètre sont des identifiants de jeux de caractères (entier long *MIBEnum* ou chaîne *Name*, identifiants définis par l'IANA, cf. http://www.iana.org/assignments/character-sets). Voici la liste des identifiants correspondant aux jeux de caractères pris en charge par le serveur Web de 4D : 4=ISO-8859-1 12=ISO-8859-9 13=ISO-8859-10 17=Shift-JIS 2024=Windows-31J 2026=Big5 38=euc-kr 106=UTF-8 2250=Windows-1250 2251=Windows-1251 2253=Windows-1253 2255=Windows-1255 2256=Windows-1256 • *Mode compatibilité ASCII* : 0 : Occidental 1 : Japonais 2 : Chinois 3 : Coréen 4 : Défini par l'utilisateur 5 : Réservé 6 : Europe Centrale 7 : Cyrillique 8 : Arabe 9 : Grec 10 : Hébreu 11 : Turc 12 : Nordique
Web niveau de compression HTTP	Entier long	50	**Portée** : Serveur Web local **Conservé entre deux sessions** : Non **Description** : Niveau de compression pour tous les échanges HTTP compressés effectués pour le serveur HTTP de 4D (requêtes client ou réponses serveur, Web et Web Service). Ce sélecteur permet d'optimiser les échanges en privilégiant la rapidité d'exécution (compression moindre) ou la quantité de compression (vitesse moindre). Le choix d'une valeur dépend de la taille et de la nature des données échangées. Passez de 1 à 9 dans le paramètre *valeur*, 1 étant la compression la plus rapide et 9 la plus élevée. Vous pouvez également passer -1 pour obtenir un compromis entre vitesse et taux de compression. Par défaut le niveau de compression est 1 (compression rapide). **Valeurs** : 1 à 9 (1 = plus rapide, 9 = plus compressé) ou -1 = meilleur compromis.
Web nom du cookie de session	Entier long	73	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de définir le nom du cookie utilisé pour stocker l'ID de session. **Valeurs :** Texte. **Valeur par défaut :** "4DSID" (passez une chaîne vide pour rétablir la valeur par défaut)
Web nombre de sessions max	Entier long	71	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description** : Permet de limiter le nombre de sessions simultanées. Lorsque le nombre défini est atteint, la session la plus ancienne est détruite (et la **Méthode base Sur fermeture process Web** est appelée) si le serveur Web a besoin d'en créer une nouvelle. **Valeurs** : Entier long. Le nombre de sessions simultanées ne peut pas dépasser le nombre total de process Web (option Web process Web simultanés maxi, 100 par défaut) **Valeur par défaut** : 100 (passez 0 pour rétablir la valeur par défaut)
Web numéro de port HTTPS	Entier long	39	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Numéro du port TCP utilisé par le serveur Web de 4D en mode local et de 4D Server pour les connexions sécurisées via SSL (protocole HTTPS). Le numéro de port HTTPS est défini dans la page "Web/Configuration" de la boîte de dialogue des Propriétés de la base. Par défaut, la valeur est 443 (valeur standard). Vous pouvez utiliser les constantes du thème **Numéros de port TCP** pour le paramètre *valeur*. **Valeurs possibles** : 0 à 65535

Constante	Type	Valeur	Comment
			Portée : 4D en mode local et 4D Server. **Conservé entre deux sessions :** Non
Web numéro du port	Entier long	15	**Description :** Permet de fixer ou de lire le numéro du port TCP utilisé par le serveur Web 4D avec 4D en mode local et 4D Server. Le numéro de port TCP est défini dans la page “Web/Configuration” de la boîte de dialogue des Propriétés de la base. Vous pouvez utiliser les constantes du thème **Numéros de port TCP** pour le paramètre *valeur*. Ce sélecteur est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d’accès au mode Développement). **Valeurs :** Pour plus d’informations sur le numéro de port TCP, reportez-vous à la section **Paramétrages du serveur Web**. **Valeur par défaut :** 80
Web process Web simultanés maxi	Entier long	18	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Limite strictement supérieure du nombre de process Web de tout type acceptés par le serveur Web avec 4D en mode local et 4D Server. Lorsque ce nombre limite (moins un) est atteint, 4D ne crée plus de nouveau process et retourne le message “Serveur non disponible” (statut HTTP 503 - Service Unavailable) à toute nouvelle requête. Ce paramètre permet de prévenir la saturation du serveur Web 4D pouvant se produire lors d’un envoi massif de requêtes ou d’une demande excessive de création de contextes. Il peut également être défini dans la boîte de dialogue des Propriétés de la base. En théorie, le nombre maximum de process Web est le résultat de la division Mémoire disponible / Taille de la pile d’un process Web. Une autre solution consiste à visualiser les informations sur les process Web affichées dans l’Exporateur d’exécution : le nombre courant de process Web et le nombre maximum atteint depuis le démarrage du serveur Web sont indiqués. **Valeurs** : Toute valeur incluse entre 10 et 32 000. La valeur par défaut est 100.
Web seuil de compression HTTP	Entier long	51	**Portée** : Serveur HTTP local **Conservé entre deux sessions** : Non **Description** : Dans le cadre d’échanges HTTP optimisés, seuil de taille de requête au-dessous de laquelle les échanges ne doivent pas être compressés. Ce paramétrage est utile pour éviter de perdre du temps machine à compresser les trop petits échanges. **Valeurs possibles** : Toute valeur de type Entier long. Le paramètre *valeur* contient une taille exprimée en octets. Par défaut, le seul de compression est fixé à 1024 octets.
Web taille max requêtes	Entier long	27	**Portée** : 4D local, 4D Server **Conservé entre deux sessions** : Oui **Description** : Taille maximale (en octets) des requêtes HTTP entrantes (POST) que le serveur Web est autorisé à traiter. Par défaut, la valeur est 2 000 000, c’est-à-dire un peu moins de 2 Mo. La valeur maximale (2 147 483 648) signifie en pratique qu’aucune limite n’est fixée. Ce paramétrage permet d’empêcher la saturation du serveur Web à cause de requêtes entrantes de trop grande taille. Lorsqu’une requête atteint la limite, le serveur Web de 4D la refuse. **Valeurs possibles** : 500 000 à 2 147 483 648.
Web timeout process	Entier long	78	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description :** Permet de modifier la durée de vie des process inactifs associés aux sessions. A l’issue du timeout, le process est tué sur le serveur, la **Méthode base Sur fermeture process Web** est appelée puis le contexte de la session est détruit. **Valeurs** : Entier long (minutes) **Valeur par défaut** : 480 minutes (passez 0 pour rétablir la valeur par défaut)
Web timeout session	Entier long	72	**Portée** : serveur Web local **Conservé entre deux sessions** : Non, mais reste valide même si le serveur HTTP est redémarré. **Description** : Permet de modifier la durée de vie des sessions inactives (durée définie dans le cookie). A l’issue de cette durée, le cookie de session expire et n’est plus envoyé par le client HTTP. **Valeurs** : Entier long (minutes) **Valeur par défaut** : 480 minutes (passez 0 pour rétablir la valeur par défaut)
Web TRACE HTTP	Entier long	85	**Portée** : serveur Web local **Conservé entre deux sessions** : Non **Description** : Permet de désactiver ou d'activer la méthode HTTP TRACE dans le serveur Web de 4D. Pour des raisons de sécurité, à compter de 4D v15 R2, par défaut le serveur Web de 4D Web rejette les requêtes HTTP TRACE avec l'erreur 405 (see Désactivation de HTTP TRACE). Si nécessaire, vous pouvez rétablir la prise en charge de la méthode HTTP TRACE pour la session en passant cette constante avec la valeur 1. Lorsque l'option est activée, le serveur Web de 4D répond aux requêtes HTTP TRACE en retournant la ligne de requête, l'en-tête et le corps. **Valeurs possibles** : 0 (désactivé) ou 1 (activé) **Valeur par défaut** : 0 (désactivé)

Signatures système standard

Les signatures Système standard sont des chaînes de 4 caractères désignant des types de fichiers standard, des types de ressources ou encore des types de données stockés notamment dans le Presse-papiers.

Constante	Type	Valeur	Comment
Document image	Chaîne	PICT	
Document MIDI Windows	Chaîne	MID	
Document son Windows	Chaîne	WAV	
Document texte	Chaîne	TEXT	
Document vidéo Windows	Chaîne	AVI	

SQL

Constante	Type	Valeur	Comment
Source de données système	Entier long	2	
Source de données utilisateur	Entier long	1	
SQL abandonner si erreur	Entier long	1	En cas d'erreur, 4D stoppe immédiatement l'exécution du script.
SQL asynchrone	Entier long	1	0 = connexion synchrone (valeur par défaut), 1 (ou valeur différente de 0) = connexion asynchrone
SQL confirmer si erreur	Entier long	2	En cas d'erreur, 4D affiche une boîte de dialogue détaillant l'erreur et permettant à l'utilisateur d'interrompre ou de poursuivre l'exécution du script.
SQL continuer si erreur	Entier long	3	En cas d'erreur, 4D l'ignore et poursuit l'exécution du script.
SQL jeu de caractères	Entier long	100	Encodage du texte utilisé pour les requêtes envoyées aux sources externes (via le SQL pass-through). La modification est effective pour le process courant et la connexion courante. Valeurs possibles : identifiant MIBEnum (cf. note 2) ou valeur -2 (cf. note 3) Par défaut : 106 (UTF-8)
SQL longueur maxi données	Entier long	3	Longueur maximale des données retournées
SQL nombre maxi lignes	Entier long	2	Nombre maximum de lignes dans l'ensemble résultant (utilisé pour les prévisualisations)
SQL paramètre entrée	Entier long	1	
SQL paramètre entrée sortie	Entier long	2	Utilisable uniquement dans le contexte d'une procédure stockée SQL (paramètre entrée-sortie défini dans la procédure stockée)
SQL paramètre sortie	Entier long	4	Utilisable uniquement dans le contexte d'une procédure stockée SQL (paramètre sortie défini dans la procédure stockée)
SQL timeout connexion	Entier long	5	Durée maximale d'attente lors de l'exécution de la commande SQL LOGIN. Cette valeur doit être fixée avant l'ouverture de la connexion pour être prise en compte Valeurs possibles : durée en secondes Par défaut : pas de timeout
SQL timeout requête	Entier long	4	Durée maximale d'attente de la réponse lors de l'exécution de la commande SQL EXECUTER. Valeurs : durée en secondes Par défaut : pas de timeout
SQL tous les enregistrements	Entier long	-1	
SQL utiliser les droits d'accès	Chaîne	SQL_Use_Access_Rights	Permet de restreindre les droits d'accès à appliquer lors de l'exécution des commandes SQL du script. Lorsque vous utilisez cet attribut, vous devez passer 0 ou 1 dans *valAttribut* : • *valAttribut* = 1 : 4D utilise les droits d'accès de l'utilisateur 4D courant. • *valAttribut* = 0 (ou attribut non défini) : 4D ne restreint pas les accès, les droits du Super_Utilisateur sont utilisés.
SQL_INTERNAL	Chaîne	;DB4D_SQL_LOCAL;	

Statut du process

Constante	Type	Valeur	Comment
Détruit	Entier long	-1	
Dialogue caché	Entier long	6	
En attente drapeau interne	Entier long	4	
En attente entrée sortie	Entier long	3	
En attente événement	Entier long	2	
En exécution	Entier long	0	
Endormi	Entier long	1	
Inexistant	Entier long	-100	
Suspendu	Entier long	5	

Styles de caractères

Les constantes préfixées _O_ sont obsolètes et ne doivent plus être utilisées.

Constante	Type	Valeur	Comment
_o_Condensé	Entier long	32	
_o_Etendu	Entier long	64	
_o_Ombré	Entier long	16	
_o_Relief	Entier long	8	
Feuille de style automatique	Chaîne	__automatic__	Utilisée par défaut pour tous les objets
Feuille de style automatique_additionnel	Chaîne	__automatic_additional_text__	Prise en charge par les textes statiques, champs et variables uniquement. Utilisée pour du texte additionnel dans les boîtes de dialogue.
Feuille de style automatique_texte principal	Chaîne	__automatic_main_text__	Prise en charge par les textes statiques, champs et variables uniquement. Utilisée pour le texte principal des boîtes de dialogue.
Gras	Entier long	1	
Gras et italique	Entier long	3	
Gras et souligné	Entier long	5	
Italique	Entier long	2	
Italique et souligné	Entier long	6	
Normal	Entier long	0	
Souligné	Entier long	4	

Texte multistyle

Constante	Type	Valeur	Comment
ST Balises comme code xml	Entier long	128	Le code XML de la balise est retourné en texte brut. Par exemple pour la balise '<img src="test.jpg" alt="image">mon image</img>', le texte brut est '<img src="test.jpg" alt="image">mon image</img>'
ST Balises comme texte brut	Entier long	64	Le libellé de la balise est retourné en texte brut. Par exemple pour la balise '<img src="test.jpg" alt="image">mon image</img>', le texte brut est "mon image" (fonctionnement par défaut dans les formulaires)
ST Début sélection	Entier long	-1000	Désigne le premier caractère de la sélection courante de texte dans l'objet (*)
ST Début texte	Entier long	1	Désigne le premier caractère du texte contenu dans l'objet
ST Expressions 4D comme sources	Entier long	2	La chaîne d'origine des références d'expressions 4D est retournée
ST Expressions 4D comme valeurs	Entier long	1	Les références d'expressions 4D sont retournées sous leur forme évaluée (fonctionnement par défaut dans les formulaires)
ST Fin sélection	Entier long	-1001	Désigne le dernier caractère de la sélection courante de texte dans l'objet (*)
ST Fin texte	Entier long	0	Désigne le dernier caractère du texte contenu dans l'objet
ST Liens utilisateur comme libellés	Entier long	16	Le libellé visible du lien utilisateur est retourné (fonctionnement par défaut dans les formulaires)
ST Liens utilisateur comme liens	Entier long	32	Le contenu du lien utilisateur est retourné
ST Mode affichage expressions	Entier long	1	Le paramètre *valeur* peut contenir ST Valeurs or ST Références
ST Références	Entier long	1	Affichage des chaînes source des expressions
ST Références comme espaces	Entier long	0	Chaque référence est retournée sous forme d'un caractère espace insécable (fonctionnement par défaut, utilisé par les autres commandes)
ST Texte visible avec Expressions 4D comme sources	Entier long	86	Retourne le texte tel qu'il est visible dans les formulaires avec la chaîne d'origine des expressions 4D. Correspond à la combinaison prédéfinie des constantes 2+4+16+64.
ST Texte visible avec Expressions 4D comme valeurs	Entier long	85	Retourne le texte tel qu'il est visible dans les formulaires avec les expressions 4D sous leur forme évaluée. Correspond à la combinaison prédéfinie de constantes 1+4+16+64.
ST Type balise inconnu	Entier long	4	La sélection contient uniquement une balise de type inconnu
ST Type brut	Entier long	0	La sélection contient du texte et aucune référence
ST Type expression	Entier long	2	La sélection contient uniquement une référence d'expression
ST Type image	Entier long	6	La sélection contient uniquement une image (zones 4D Write Pro uniquement)
ST Type mixte	Entier long	3	La sélection contient au moins deux types de contenus différents
ST Type Url	Entier long	1	La sélection contient uniquement une référence d'URL
ST Type utilisateur	Entier long	5	La sélection contient uniquement une référence personnalisée
ST URL comme libellés	Entier long	4	Le libellé visible des URLs est retourné, par exemple "Visitez notre site Web" (fonctionnement par défaut dans les formulaires)
ST URL comme liens	Entier long	8	Le lien est retourné, par exemple "http://www.4d.com"
ST Valeurs	Entier long	0	Affichage des valeurs calculées des expressions

Touches de fonction

Constante	Type	Valeur	Comment
Touche aide	Entier long	5	
Touche bas	Entier long	31	
Touche début	Entier long	1	
Touche droite	Entier long	29	
Touche échappement	Entier long	27	
Touche entrée	Entier long	3	
Touche F1	Entier long	-122	
Touche F10	Entier long	-109	
Touche F11	Entier long	-103	
Touche F12	Entier long	-111	
Touche F13	Entier long	-105	
Touche F14	Entier long	-107	
Touche F15	Entier long	-113	
Touche F2	Entier long	-120	
Touche F3	Entier long	-99	
Touche F4	Entier long	-118	
Touche F5	Entier long	-96	
Touche F6	Entier long	-97	
Touche F7	Entier long	-98	
Touche F8	Entier long	-100	
Touche F9	Entier long	-101	
Touche fin	Entier long	4	
Touche gauche	Entier long	28	
Touche haut	Entier long	30	
Touche page précédente	Entier long	12	
Touche page suivante	Entier long	11	
Touche retour arrière	Entier long	8	
Touche retour chariot	Entier long	13	
Touche tab	Entier long	9	

Touches équivalents clavier

Constante	Type	Valeur	Comment
Raccourci avec Aide	Chaîne	[help]	
Raccourci avec Début	Chaîne	[home]	
Raccourci avec Echappement	Chaîne	[esc]	
Raccourci avec Effacement arrière	Chaîne	[backspace]	
Raccourci avec Entrée	Chaîne	[enter]	
Raccourci avec F1	Chaîne	[F1]	
Raccourci avec F10	Chaîne	[F10]	
Raccourci avec F11	Chaîne	[F11]	
Raccourci avec F12	Chaîne	[F12]	
Raccourci avec F13	Chaîne	[F13]	
Raccourci avec F14	Chaîne	[F14]	
Raccourci avec F15	Chaîne	[F15]	
Raccourci avec F2	Chaîne	[F2]	
Raccourci avec F3	Chaîne	[F3]	
Raccourci avec F4	Chaîne	[F4]	
Raccourci avec F5	Chaîne	[F5]	
Raccourci avec F6	Chaîne	[F6]	
Raccourci avec F7	Chaîne	[F7]	
Raccourci avec F8	Chaîne	[F8]	
Raccourci avec F9	Chaîne	[F9]	
Raccourci avec Fin	Chaîne	[end]	
Raccourci avec Flèche bas	Chaîne	[down arrow]	
Raccourci avec Flèche droite	Chaîne	[right arrow]	
Raccourci avec Flèche gauche	Chaîne	[left arrow]	
Raccourci avec Flèche haut	Chaîne	[up arrow]	
Raccourci avec Page préc	Chaîne	[page up]	
Raccourci avec Page suiv	Chaîne	[page down]	
Raccourci avec Retour Chariot	Chaîne	[return]	
Raccourci avec Suppression	Chaîne	[del]	
Raccourci avec Tabulation	Chaîne	[tab]	

Transformation des images

Constante	Type	Valeur	Comment
Concaténation horizontale	Entier long	1	
Concaténation verticale	Entier long	2	
Miroir horizontal	Entier long	3	
Miroir vertical	Entier long	4	
Passage en niveaux de gris	Entier long	101	
Recadrage	Entier long	100	
Redimensionnement	Entier long	1	
Réinitialisation	Entier long	0	
Superposition	Entier long	3	
Translation	Entier long	2	
Transparence	Entier long	102	

Type de liste des polices

Constante	Type	Valeur	Comment
Polices favorites	Entier long	1	*polices* contient la liste des polices favorites. - Sous Windows : liste des noms de famille des polices actives. - Sous OS X : liste des noms de famille des polices présente dans le panneau de configuration nommé "Favorites" en anglais, "Favoris" en français", "Favoriten" en allemand, etc. Cette collection peut être vide si l'utilisateur n'a ajouté aucune police favorite.
Polices récentes	Entier long	2	*polices* contient la liste des polices récentes (liste des polices utilisées lors de la session 4D). Cette liste est notamment utilisée par les zones de texte multistyle.
Polices système	Entier long	0	*polices* contient la liste de toutes les polices système. Option par défaut si *typeListe* est omis.

Type digest

Constante	Type	Valeur	Comment
Digest 4D	Entier long	2	Utiliser l'algorithme interne de 4D
Digest MD5	Entier long	0	Utiliser l'algorithme MD5
Digest SHA1	Entier long	1	Utiliser l'algorithme SHA-1

Type du process

Constante	Type	Valeur	Comment
_o_Process Web avec contexte	Entier long	-11	
Aucun	Entier long	0	
Autre process 4D	Entier long	-10	
Autre process utilisateur	Entier long	4	
Créé par commande de menu	Entier long	2	
Créé par dialogue d'exécution	Entier long	3	
Gestionnaire Apple Event	Entier long	-7	
Gestionnaire d'événement	Entier long	-8	
Gestionnaire d'index	Entier long	-5	
Gestionnaire du cache	Entier long	-4	
Gestionnaire du port série	Entier long	-6	
Process 4D Server interne	Entier long	-18	
Process CSM	Entier long	-22	
Process d'activité	Entier long	-26	
Process de restitution	Entier long	-21	
Process de sauvegarde	Entier long	-19	
Process développement	Entier long	-2	
Process du fichier d'historique	Entier long	-20	
Process du serveur Web	Entier long	-13	
Process exécuté sur client	Entier long	-14	
Process exécuté sur serveur	Entier long	1	
Process exécution méthode SQL	Entier long	-24	
Process gestionnaire de clients	Entier long	-31	
Process interface serveur	Entier long	-15	
Process macro éditeur de méthod	Entier long	-17	
Process minuteur interne	Entier long	-25	
Process principal	Entier long	-1	
Process sur fermeture	Entier long	-16	
Process Web 4D distant	Entier long	-12	
Process Web sans contexte	Entier long	-3	
Process worker	Entier long	5	Process worker lancé par l'utilisateur
Tâche externe	Entier long	-9	

Type index

Constante	Type	Valeur	Comment
Index BTree cluster	Entier long	3	Index de type B-Tree utilisant des clusters. Ce type d'index est optimisé lorsque l'index contient peu de clés, c'est-à-dire lorsque les mêmes valeurs reviennent souvent dans les données.
Index BTree standard	Entier long	1	Index de type B-Tree classique. Ce type d'index polyvalent est utilisé dans les versions précédentes de 4D
Index de mots clés	Entier long	-1	Permet l'indexation mot à mot du contenu du champ. Ce type d'index est utilisable avec les champs de type Texte, Alpha et Image. Attention, les index de mots-clés ne peuvent pas être composites.
Type index par défaut	Entier long	0	4D définit le type d'index (hors index de mots-clés) le plus optimisé en fonction du contenu du champ.

Types champs et variables

Constante	Type	Valeur	Comment
Est un BLOB	Entier long	30	
Est un booléen	Entier long	6	
Est un champ alpha	Entier long	0	
Est un entier	Entier long	8	
Est un entier 64 bits	Entier long	25	
Est un entier long	Entier long	9	
Est un float	Entier long	35	
Est un null JSON	Entier long	255	
Est un numérique	Entier long	1	
Est un objet	Entier long	38	
Est un pointeur	Entier long	23	
Est un tableau 2D	Entier long	13	
Est un tableau blob	Entier long	31	
Est un tableau booléen	Entier long	22	
Est un tableau chaîne	Entier long	21	
Est un tableau date	Entier long	17	
Est un tableau entier	Entier long	15	
Est un tableau entierlong	Entier long	16	
Est un tableau heure	Entier long	32	
Est un tableau image	Entier long	19	
Est un tableau numérique	Entier long	14	
Est un tableau objet	Entier long	39	
Est un tableau pointeur	Entier long	20	
Est un tableau texte	Entier long	18	
Est un texte	Entier long	2	
Est une date	Entier long	4	
Est une heure	Entier long	11	
Est une image	Entier long	3	
Est une sous table	Entier long	7	
Est une variable chaîne	Entier long	24	
Est une variable indéfinie	Entier long	5	

Types objets formulaire

Constante	Type	Valeur	Comment
Objet type bouton 3D	Entier long	16	
Objet type bouton image	Entier long	19	
Objet type bouton inversé	Entier long	17	
Objet type bouton invisible	Entier long	18	
Objet type bouton poussoir	Entier long	15	
Objet type bouton radio	Entier long	22	
Objet type bouton radio 3D	Entier long	23	
Objet type bouton radio image	Entier long	24	
Objet type cadran	Entier long	28	
Objet type case à cocher	Entier long	25	
Objet type case à cocher 3D	Entier long	26	
Objet type champ radio boutons	Entier long	5	
Objet type combobox	Entier long	11	
Objet type grille de boutons	Entier long	20	
Objet type groupe	Entier long	21	
Objet type image statique	Entier long	2	
Objet type inconnu	Entier long	0	
Objet type indicateur de progression	Entier long	27	
Objet type ligne	Entier long	32	
Objet type listbox	Entier long	7	
Objet type listbox colonne	Entier long	9	
Objet type listbox entête	Entier long	8	
Objet type listbox pied	Entier long	10	
Objet type liste hiérarchique	Entier long	6	
Objet type matrice	Entier long	35	
Objet type menu déroulant hiérarchique	Entier long	13	
Objet type onglet	Entier long	37	
Objet type ovale	Entier long	34	
Objet type popup liste déroulante	Entier long	12	
Objet type popup menu image	Entier long	14	
Objet type rectangle	Entier long	31	
Objet type rectangle arrondi	Entier long	33	
Objet type règle	Entier long	29	
Objet type saisie image	Entier long	4	
Objet type saisie texte	Entier long	3	
Objet type séparateur	Entier long	36	
Objet type sous-formulaire	Entier long	39	
Objet type texte statique	Entier long	1	
Objet type zone de groupe	Entier long	30	
Objet type zone plug-in	Entier long	38	
Objet type zone web	Entier long	40	
Objet type zone write pro	Entier long	41	

Valeurs des métadonnées images

Constante	Type	Valeur	Comment
EXIF Action	Entier long	6	
EXIF Adobe RGB	Entier long	2	
EXIF Aperture Priority AE	Entier long	3	
EXIF Auto	Entier long	0	
EXIF Auto Bracket	Entier long	2	
EXIF Auto Mode	Entier long	3	
EXIF Average	Entier long	1	
EXIF B	Entier long	6	
EXIF Cb	Entier long	2	
EXIF Center Weighted Average	Entier long	2	
EXIF Close	Entier long	2	
EXIF Cloudy	Entier long	10	
EXIF Color Sequential Area	Entier long	5	
EXIF Color Sequential Linear	Entier long	8	
EXIF Compulsory Flash Firing	Entier long	1	
EXIF Compulsory Flash Suppression	Entier long	2	
EXIF Cool White Fluorescent	Entier long	14	
EXIF Cr	Entier long	3	
EXIF Creative	Entier long	5	
EXIF Custom	Entier long	1	
EXIF D50	Entier long	23	
EXIF D55	Entier long	20	
EXIF D65	Entier long	21	
EXIF D75	Entier long	22	
EXIF Day White Fluorescent	Entier long	13	
EXIF Daylight	Entier long	1	
EXIF Daylight Fluorescent	Entier long	12	
EXIF Detected	Entier long	3	
EXIF Digital Camera	Entier long	3	
EXIF Distant	Entier long	3	
EXIF Exposure Portrait	Entier long	7	
EXIF Film Scanner	Entier long	1	
EXIF Fine Weather	Entier long	9	
EXIF Flashlight	Entier long	4	
EXIF G	Entier long	5	
EXIF High	Entier long	2	
EXIF High Gain Down	Entier long	4	
EXIF High Gain Up	Entier long	2	
EXIF ISOStudio Tungsten	Entier long	24	
EXIF Landscape	Entier long	8	
EXIF Light Fluorescent	Entier long	2	
EXIF Low	Entier long	1	
EXIF Low Gain Down	Entier long	3	
EXIF Low Gain Up	Entier long	1	
EXIF Macro	Entier long	1	
EXIF Manual	Entier long	1	
EXIF Multi Segment	Entier long	5	
EXIF Multi Spot	Entier long	4	
EXIF Night	Entier long	3	
EXIF No Detection Function	Entier long	0	
EXIF None	Entier long	0	
EXIF Normal	Entier long	0	
EXIF Not Defined	Entier long	1	
EXIF Not Detected	Entier long	2	
EXIF One Chip Color Area	Entier long	2	
EXIF Other	Entier long	255	
EXIF Partial	Entier long	6	
EXIF Program AE	Entier long	2	
EXIF R	Entier long	4	
EXIF Reflection Print Scanner	Entier long	2	
EXIF Reserved	Entier long	1	
EXIF s RGB	Entier long	1	
EXIF Scene Landscape	Entier long	1	
EXIF Scene Portrait	Entier long	2	
EXIF Shade	Entier long	11	
EXIF Shutter Speed Priority AE	Entier long	4	
EXIF Spot	Entier long	3	

Constante	Type	Valeur	Comment
EXIF Standard	Entier long	0	
EXIF Standard Light A	Entier long	17	
EXIF Standard Light B	Entier long	18	
EXIF Standard Light C	Entier long	19	
EXIF Three Chip Color Area	Entier long	4	
EXIF Trilinear	Entier long	7	
EXIF Tungsten	Entier long	3	
EXIF Two Chip Color Area	Entier long	3	
EXIF Uncalibrated	Entier long	-1	
EXIF Unknown	Entier long	0	
EXIF Unused	Entier long	0	
EXIF White Fluorescent	Entier long	15	
EXIF Y	Entier long	1	
GPS 2D	Entier long	2	
GPS 3D	Entier long	3	
GPS Above Sea Level	Entier long	0	
GPS Below Sea Level	Entier long	1	
GPS Correction Applied	Entier long	1	
GPS Correction Not Applied	Entier long	0	
GPS East	Chaîne	E	
GPS km h	Chaîne	K	
GPS knots h	Chaîne	K	
GPS Magnetic north	Chaîne	M	
GPS Measurement in progress	Chaîne	A	
GPS Measurement Interoperability	Chaîne	V	
GPS miles h	Chaîne	M	
GPS North	Chaîne	N	
GPS South	Chaîne	S	
GPS True north	Chaîne	T	
GPS West	Chaîne	W	
IPTC Action	Entier long	11900	
IPTC Aerial View	Entier long	11200	
IPTC Close Up	Entier long	11800	
IPTC Couple	Entier long	10700	
IPTC Exterior View	Entier long	11600	
IPTC Full Length	Entier long	10300	
IPTC General View	Entier long	11000	
IPTC Group	Entier long	10900	
IPTC Half Length	Entier long	10200	
IPTC Headshot	Entier long	10100	
IPTC Interior View	Entier long	11700	
IPTC Movie Scene	Entier long	12400	
IPTC Night Scene	Entier long	11400	
IPTC Off Beat	Entier long	12300	
IPTC Panoramic View	Entier long	11100	
IPTC Performing	Entier long	12000	
IPTC Posing	Entier long	12100	
IPTC Profile	Entier long	10400	
IPTC Rear View	Entier long	10500	
IPTC Satellite	Entier long	11500	
IPTC Single	Entier long	10600	
IPTC Symbolic	Entier long	12200	
IPTC Two	Entier long	10800	
IPTC Under Water	Entier long	11300	
TIFF Adobe Deflate	Entier long	8	
TIFF Black Is Zero	Entier long	1	
TIFF CCIRLEW	Entier long	32771	
TIFF CCITT1D	Entier long	2	
TIFF CIELab	Entier long	8	
TIFF CM	Entier long	3	
TIFF CMYK	Entier long	5	
TIFF Color Filter Array	Entier long	32803	
TIFF DCS	Entier long	32947	
TIFF Deflate	Entier long	32946	
TIFF Epson ERF	Entier long	32769	
TIFF Horizontal	Entier long	1	
TIFF ICCLab	Entier long	9	
TIFF Inches	Entier long	2	
TIFF IT8BL	Entier long	32898	
TIFF IT8CTPAD	Entier long	32895	

Constante	Type	Valeur	Comment
TIFF IT8LW	Entier long	32896	
TIFF IT8MP	Entier long	32897	
TIFF ITULab	Entier long	10	
TIFF JBIG	Entier long	34661	
TIFF JBIGB&W	Entier long	9	
TIFF JBIGColor	Entier long	10	
TIFF JPEG	Entier long	7	
TIFF JPEG2000	Entier long	34712	
TIFF JPEGThumbs Only	Entier long	6	
TIFF Kodak DCR	Entier long	65000	
TIFF Kodak KDC	Entier long	32867	
TIFF Kodak262	Entier long	262	
TIFF Linear Raw	Entier long	34892	
TIFF LZW	Entier long	5	
TIFF MDIBinary Level Codec	Entier long	34718	
TIFF MDIProgressive Transform Codec	Entier long	34719	
TIFF MDIVector	Entier long	34720	
TIFF Mirror Horizontal	Entier long	2	
TIFF Mirror Horizontal And Rotate270CW	Entier long	5	
TIFF Mirror Horizontal And Rotate90CW	Entier long	7	
TIFF Mirror Vertical	Entier long	4	
TIFF MM	Entier long	4	
TIFF Next	Entier long	32766	
TIFF Nikon NEF	Entier long	34713	
TIFF None	Entier long	1	
TIFF Pack Bits	Entier long	32773	
TIFF Pentax PEF	Entier long	65535	
TIFF Pixar Film	Entier long	32908	
TIFF Pixar Log	Entier long	32909	
TIFF Pixar Log L	Entier long	32844	
TIFF Pixar Log Luv	Entier long	32845	
TIFF RGB	Entier long	2	
TIFF RGBPalette	Entier long	3	
TIFF Rotate180	Entier long	3	
TIFF Rotate270CW	Entier long	8	
TIFF Rotate90CW	Entier long	6	
TIFF SGILog	Entier long	34676	
TIFF SGILog24	Entier long	34677	
TIFF Sony ARW	Entier long	32767	
TIFF T4Group3Fax	Entier long	3	
TIFF T6Group4Fax	Entier long	4	
TIFF Thunderscan	Entier long	32809	
TIFF Transparency Mask	Entier long	4	
TIFF UM	Entier long	5	
TIFF Uncompressed	Entier long	1	
TIFF White Is Zero	Entier long	0	
TIFF YCb Cr	Entier long	6	

Valeurs pour Actions standard associée

Toutes les constantes de ce thème sont obsolètes à compter de 4D v16 R3. Il est désormais recommandé d'utiliser les constantes du thème **Valeurs Texte pour Action standard associée**.

Constante	Type	Valeur	Comment
Action afficher Presse papiers	Entier long	23	
Action ajouter sous enreg	Entier long	14	
Action annuler	Entier long	17	
Action coller	Entier long	20	
Action copier	Entier long	19	
Action couper	Entier long	18	
Action CSM	Entier long	36	
Action dernier enregistrement	Entier long	6	
Action dernière page	Entier long	11	
Action effacer	Entier long	21	
Action enregistrement précédent	Entier long	4	
Action enregistrement suivant	Entier long	3	
Action modifier sous enreg	Entier long	12	
Action ne pas valider	Entier long	1	
Action page précédente	Entier long	9	
Action page suivante	Entier long	8	
Action premier enregistrement	Entier long	5	
Action première page	Entier long	10	
Action propriétés de la base	Entier long	32	
Action quitter	Entier long	27	
Action répéter	Entier long	31	
Action supprimer enregistrement	Entier long	7	
Action supprimer sous enreg	Entier long	13	
Action test application	Entier long	26	
Action tout sélectionner	Entier long	22	
Action valider	Entier long	2	
Pas d'action	Entier long	0	
Retour au mode Développement	Entier long	35	

Valeurs Texte pour Action standard associée

Constante	Type	Valeur	Comment
Objet action Actualiser URL courant	Chaîne	39	
Objet action Afficher Presse papiers	Chaîne	23	
Objet action Ajouter sous enreg	Chaîne	14	
Objet action Aller à page	Chaîne	15	
Objet action Annuler	Chaîne	1	
Objet action Annuler édition	Chaîne	17	
Objet action Arrêter chargement URL	Chaîne	40	
Objet action Coller	Chaîne	20	
Objet action Copier	Chaîne	19	
Objet action Couper	Chaîne	18	
Objet action CSM	Chaîne	36	
Objet action Dernier enreg	Chaîne	6	
Objet action Dernière page	Chaîne	11	
Objet action Effacer	Chaîne	21	
Objet action Enreg précédent	Chaîne	4	
Objet action Enreg suivant	Chaîne	3	
Objet action Modifier sous enreg	Chaîne	12	
Objet action Ouvrir URL précédent	Chaîne	37	
Objet action Ouvrir URL suivant	Chaîne	38	
Objet action Page précédente	Chaîne	9	
Objet action Page suivante	Chaîne	8	
Objet action Premier enreg	Chaîne	5	
Objet action Première page	Chaîne	10	
Objet action Propriétés de la base	Chaîne	32	
Objet action Quitter	Chaîne	27	
Objet action Répéter	Chaîne	31	
Objet action Retour au mode Développement	Chaîne	35	
Objet action Séparateur auto	Chaîne	16	
Objet action Supprimer enreg	Chaîne	7	
Objet action Supprimer sous enreg	Chaîne	13	
Objet action Test application	Chaîne	26	
Objet action Tout sélectionner	Chaîne	22	
Objet action Valider	Chaîne	2	
Objet sans action standard	Chaîne	0	

Web Services (Client)

Constante	Type	Valeur	Comment
Web Service afficher dial auth	Entier long	4	*valeur* = 0 (ne pas afficher le dialogue) ou 1 (afficher le dialogue) Cette option gère l'affichage de boîte de dialogue d'authentification lors de l'exécution de la commande **WEB SERVICE APPELER**. Par défaut, cette commande ne provoque jamais l'affichage de la boîte de dialogue, vous devez en principe utiliser la commande **WEB SERVICE AUTHENTIFIER**. Toutefois, si vous souhaitez qu'une boîte de dialogue d'authentification apparaisse pour que l'utilisateur saisisse ses identifiants, vous devez utiliser cette option : passez 1 dans *valeur* pour afficher la boîte de dialogue, et 0 sinon. La boîte de dialogue n'apparaît que si le service Web requiert une authentification.
Web Service code erreur	Entier long	0	Code d'erreur principal (défini par 4D). Ce code est également retourné dans la variable système Error. Voici la liste des codes pouvant être retournés : 9910 : Erreur Web Service (voir aussi Web Service origine erreur) 9911 : Erreur de l'analyseur xml 9912 : Erreur HTTP (voir aussi Web Service code erreur HTTP) 9913 : Erreur réseau 9914 : Erreur interne
Web Service code statut HTTP	Entier long	2	Code de statut HTTP (à utiliser en cas d'erreur principale 9912).
Web Service compression	Entier long	1	
Web Service compression HTTP	Entier long	6	*valeur* = Web Service compression Cette option permet d'activer un mécanisme interne de compression des requêtes SOAP afin d'accélérer les échanges inter-applications 4D. Lorsque vous exécutez l'instruction **WEB SERVICE FIXER OPTION**(Web Service compression HTTP; Web Service compression) sur le client 4D du Web Service, les données de la prochaine requête SOAP envoyée par le client seront compressées en utilisant un mécanisme standard HTTP ("gzip" ou "deflate" en fonction du contenu de la requête) avant leur envoi au serveur SOAP 4D. Le serveur décompressera et analysera la requête puis répondra en utilisant automatiquement le même mécanisme. Seule la requête suivant l'appel de la commande **WEB SERVICE FIXER OPTION** est affectée. Vous devez donc appeler cette commande chaque fois que vous voulez utiliser la compression. Par défaut, 4D ne compresse pas les requêtes HTTP des Web Services. **Note :** Ce mécanisme ne peut pas être utilisé pour des requêtes adressées à un serveur SOAP 4D d'une version antérieure à la 11.3. Afin de vous permettre d'optimiser encore ce fonctionnement, des options supplémentaires configurent le seuil et le taux de compression des requêtes. Ces options sont accessibles via la commande **FIXER PARAMETRE BASE**.
Web Service dynamique	Entier long	0	
Web Service effacer infos auth	Entier long	5	*valeur* = 0 (ne pas effacer les informations) ou 1 (les effacer) Cette option permet d'indiquer à 4D de mémoriser les informations d'authentification de l'utilisateur (nom d'utilisateur, mot de passe, méthode, etc.), dans le but de les réutiliser par la suite. Par défaut, ces informations sont effacées après chaque exécution de la commande **WEB SERVICE APPELER**. Passez 0 dans *valeur* pour les mémoriser et 1 pour les effacer. A noter que lorsque vous passez 0, les informations sont conservées pendant la session mais ne sont pas stockées.
Web Service entrée manuel	Entier long	1	
Web Service header SOAP	Entier long	2	*valeur* = référence d'élément xml racine à insérer en tant que header (en-tête) de la requête SOAP. Cette option permet d'insérer un header dans la requête SOAP générée par la commande **WEB SERVICE APPELER**. Par défaut, les requêtes SOAP ne comportent pas d'en-tête spécifique. Cependant, certains Web Services requièrent la présence de cet en-tête, par exemple pour la gestion de paramètres d'identification.
Web Service manuel	Entier long	3	
Web Service message	Entier long	1	Message détaillé décrivant l'erreur. Le type de message diffère suivant le type d'erreur principale. - Si erreur principale = 9910 (Erreur Web Service) : la cause de l'erreur SOAP est retournée (ex : "méthode appelée inexistante"). - Si erreur principale = 9911 (Erreur de l'analyseur xml) : l'emplacement de l'erreur dans le document xml est retourné. - Si erreur principale = 9912 (Erreur HTTP) : - si l'erreur HTTP est située dans l'intervalle [300-400] (problèmes lié à l'emplacement du document demandé), le nouvel emplacement de l'URL demandé est retourné. - pour tout autre code d'erreur HTTP, le <body> est renvoyé. - Si erreur principale = 9913 (Erreur réseau) : la cause de l'erreur réseau est retournée (ex : "AdresseServeur : erreur DNS") - Si erreur principale = 9914 (Erreur interne) : la cause de l'erreur interne est retournée
Web Service origine erreur	Entier long	3	Cause de l'erreur (retournée par le protocole SOAP — à utiliser en cas d'erreur principale 9910). - Version Mismatch (les versions ne correspondent pas). - Must Understand (un paramètre défini comme obligatoire n'a pas pu être interprété par le serveur) - Sender Fault - Receiver Fault - Encoding Unknown (encodage inconnu)
Web Service SOAP_1_1	Entier long	0	
Web Service SOAP_1_2	Entier long	1	
Web Service sortie manuel	Entier long	2	
Web Service timeout HTTP	Entier long	1	*valeur* = "timeout" de la partie cliente exprimé en secondes. Le timeout de la partie cliente est le délai d'attente du client Web Service en cas de non-réponse du serveur. A l'issue de ce délai, le client referme la session, la requête est perdue. Par défaut, ce délai est de 180 secondes. Il peut être modifié en raison de caractéristiques particulières (état du réseau, spécificités du Web Service, etc.).

Constante	Type	Valeur	Commentaire
Web Service version SOAP	Entier long	3	Valeur : Web Service SOAP_1_1 ou Web Service SOAP_1_2 Cette option permet de préciser la version du protocole SOAP utilisée dans la requête. Passez dans valeur la constante Web Service SOAP_1_1 pour indiquer la version 1.1 et la constante Web Service SOAP_1_2 pour indiquer la version 1.2.

Web Services (Serveur)

Constante	Type	Valeur	Comment
Est un XML	Entier long	36	
Est une référence DOM	Entier long	37	
SOAP entrée	Entier long	1	
SOAP erreur client	Entier long	1	
SOAP erreur serveur	Entier long	2	
SOAP nom méthode	Entier long	1	Nom de la méthode offerte comme Web Service sur le point d'être exécutée
SOAP nom service	Entier long	2	Nom du Web Service auquel appartient la méthode
SOAP sortie	Entier long	2	

XML

Constante	Type	Valeur	Comment
Copier source données XML	Entier long	1	4D conserve une copie de l'arbre DOM avec l'image, ce qui permet de la sauvegarder dans un champ image de la base de données et de la réafficher ou de l'exporter à tout moment.
Encoding	Entier long	4	Encodage utilisé (UTF-8, ISO...)
ID PUBLIC	Entier long	1	Identificateur public (FPI) de la DTD à laquelle le document se conforme (si la balise DOCTYPE xxx PUBLIC est présente)
ID SYSTEM	Entier long	2	Identificateur système
Lire source données XML	Entier long	0	4D lit uniquement la source de données XML, elle n'est pas conservée avec l'image. Ce paramétrage accélère sensiblement l'exécution de la commande, toutefois l'arbre DOM n'étant pas conservé, il ne sera pas possible de stocker ni d'exporter l'image.
Nom DOCTYPE	Entier long	3	Nom de l'élément racine tel que défini dans la balise DOCTYPE
Posséder source données XML	Entier long	2	4D exporte l'arbre DOM avec l'image. L'image pourra être stockée ou exportée et l'exécution de la commande est rapide. Toutefois, la référence XML *refElément* n'est alors plus utilisable par les autres commandes 4D. Ce mode d'exportation est utilisé par défaut si le paramètre *typeExport* est omis.
URI document	Entier long	6	URI de la DTD
Version	Entier long	5	Version de XML accepté
XML avec échappement	Entier long	1	
XML avec indentation	Entier long	1	
XML Base64	Entier long	1	
XML CDATA	Entier long	7	
XML codec natif	Entier long	2	
XML commentaire	Entier long	2	
XML convertir en PNG	Entier long	1	
XML DATA	Entier long	6	
XML data URI scheme	Entier long	2	
XML datetime local	Entier long	3	
XML datetime local absolu	Entier long	1	
XML datetime UTC	Entier long	5	
XML début document	Entier long	1	
XML début élément	Entier long	4	
XML DOCTYPE	Entier long	10	
XML données brutes	Entier long	2	
XML durée	Entier long	2	
XML Elément	Entier long	11	
XML encodage binaire	Entier long	5	Définit la manière dont les données binaires seront converties. **Valeurs possibles :** • XML Base64 (valeur par défaut) : les données binaires sont simplement converties en base64. • XML data URI scheme : les données binaires sont converties en base64 et l'en-tête "data:;base64" est ajouté. Ce format permet principalement à un navigateur de décoder automatiquement une image, et est également requis pour l'insertion d'images . Pour plus d'informations, voir http://en.wikipedia.org/wiki/Data_URI_scheme.
XML encodage chaînes	Entier long	1	Définit la manière dont les chaînes 4D sont converties en valeurs d'éléments. Il ne concerne pas les conversions en attributs pour lesquelles le XML impose l'utilisation de caractères d'échappement. **Valeurs possibles :** • XML avec échappement (valeur par défaut) : conversion des chaînes 4D en valeurs d'éléments XML avec remplacement des caractères. Les données de type texte sont automatiquement analysées de manière à ce que les caractères interdits (<&>') soient remplacés par des entités XML (&<> '"). • XML données brutes : les chaînes 4D sont envoyées en tant que données brutes, 4D n'effectue ni encodage ni analyse. Les valeurs 4D seront converties si possible en fragments XML et insérées en tant qu'enfant de l'élément

Constante	Type	Valeur	Comment
			cible. Si une valeur ne peut pas être considérée comme fragment XML, elle est insérée sous forme de donnée brute dans un nouveau noeud CDATA
XML encodage dates	Entier long	2	Définit la manière dont les dates 4D seront converties. Par exemple, le !01/01/2003! dans le fuseau horaire de Paris. **Valeurs possibles :** • XML ISO (valeur par défaut) : utilisation du format xs:datetime sans indication de fuseau horaire. Résultat : "2003-01-01". La partie heure, si elle est présente dans la valeur 4D (via le SQL), est perdue. • XML local : utilisation du format xs:date avec indication de fuseau horaire. Résultat : "2003-01-01 +01:00". La partie heure, si elle est présente dans la valeur 4D (via le SQL), est perdue. • XML datetime local : utilisation du format xs:dateTime (ISO 8601). Indication de fuseau horaire. Ce format permet de conserver la partie heure, si elle est présente dans la valeur 4D (via le SQL). Résultat : "<Date>2003-01-01T00:00:00 +01:00</Date>". • XML UTC : utilisation du format xs:date. Résultat : "2003-01-01Z". La partie heure, si elle est présente dans la valeur 4D (via le SQL), est perdue. • XML datetime UTC : utilisation du format xs:dateTime (ISO 8601). Ce format permet de conserver la partie heure, si elle est présente dans la valeur 4D (via le SQL). Résultat : "<Date>2003-01-01T00:00:00Z</Date>".
XML encodage heures	Entier long	3	Définit la manière dont les heures 4D seront converties. Par exemple, ?02/00/46? (heure de Paris). L'encodage diffère suivant que vous souhaitez exprimer une heure ou une durée. **Valeurs possibles pour les heures :** • XML datetime UTC : heure exprimée en UTC (Temps Universel Coordonné). A noter que la conversion en UTC est automatique. Résultat : "<Duree>0000-00-00T01:00:46Z</Duree>". • XML datetime local : heure exprimée avec le décalage horaire de la machine du moteur de 4D. Résultat : "<Duree>0000-00-00T02:00:46+01:00</Duree>". • XML datetime local absolu (valeur par défaut) : heure exprimée sans indication de fuseau horaire. Pas de modification de la valeur. Résultat : "<Duree>0000-00-00T02:00:46</Duree>". **Valeurs possibles pour les durées :** • XML secondes : nombre de secondes depuis minuit, pas de modification de la valeur puisqu'elle exprime une durée. Résultat : "<Duree>7246</Duree>". • XML durée : durée exprimée de manière conforme au XML Schema Part 2: Datatypes Second Edition. Pas de modification de la valeur puisqu'elle exprime une durée. Résultat : "<Duree>PT02H00M46S</Duree>".
XML encodage images	Entier long	6	Définit la manière dont les images doivent converties (avant l'encodage en base64). **Valeurs possibles :** • XML convertir en PNG (valeur par défaut) : les images sont converties en PNG avant d'être encodées en base64. • XML codec natif : les images sont converties dans leur premier CODEC natif de stockage avant d'être encodées en base64. Vous devez utiliser ces options pour encoder des images SVG (voir exemple de la commande **XML FIXER OPTIONS**).
XML entité	Entier long	8	
XML fin document	Entier long	9	
XML fin élément	Entier long	5	
XML indentation	Entier long	4	Définit l'indentation du *document* XML. **Valeurs possibles :** • XML avec indentation (valeur par défaut) : le document est indenté. • XML sans indentation : le document n'est pas indenté, son contenu est placé sur une seule ligne.
XML instruction de traitement	Entier long	3	
XML ISO	Entier long	1	
XML local	Entier long	2	
XML sans indentation	Entier long	2	
XML secondes	Entier long	4	
XML UTC	Entier long	4	

Zone de formulaire

Constante	Type	Valeur	Comment
Corps formulaire	Entier long	0	
Entête formulaire	Entier long	200	
Entête formulaire1	Entier long	201	
Entête formulaire10	Entier long	210	
Entête formulaire2	Entier long	202	
Entête formulaire3	Entier long	203	
Entête formulaire4	Entier long	204	
Entête formulaire5	Entier long	205	
Entête formulaire6	Entier long	206	
Entête formulaire7	Entier long	207	
Entête formulaire8	Entier long	208	
Entête formulaire9	Entier long	209	
Pied de page formulaire	Entier long	100	
Rupture formulaire0	Entier long	300	
Rupture formulaire1	Entier long	301	
Rupture formulaire2	Entier long	302	
Rupture formulaire3	Entier long	303	
Rupture formulaire4	Entier long	304	
Rupture formulaire5	Entier long	305	
Rupture formulaire6	Entier long	306	
Rupture formulaire7	Entier long	307	
Rupture formulaire8	Entier long	308	
Rupture formulaire9	Entier long	309	

Zone Web

Constante	Type	Valeur	Comment
WA autoriser applets Java	Entier long	1	Autoriser l'exécution d'applets Java dans la zone Web
WA autoriser déposer URL	Entier long	101	Autoriser le déposer d'URLs ou de fichiers dans la zone Web (défaut = Faux)
WA autoriser inspecteur Web	Entier long	100	Autoriser l'affichage de l'inspecteur Web dans la zone
WA autoriser JavaScript	Entier long	2	Autoriser l'exécution de code JavaScript dans la zone Web
WA autoriser menu contextuel	Entier long	4	Autoriser l'affichage du menu contextuel standard dans la zone Web
WA autoriser plugins	Entier long	3	Autoriser l'installation de plug-ins dans la zone Web
WA URLs précédents	Entier long	0	
WA URLs suivants	Entier long	1	

Codes d'erreurs

Erreurs de syntaxe (1 -> 81)

Le tableau suivant liste les codes et les messages des erreurs de syntaxe qui peuvent survenir lors de l'exécution de votre code en mode Développement ou Application. Quelques erreurs peuvent se produire en mode interprété seulement, quelques-unes en mode compilé seulement et les autres dans les deux modes. Ces erreurs peuvent être interceptées par une méthode d'appel sur erreur installée par la commande **APPELER SUR ERREUR**.

Code	Description
1	Il manque une parenthèse ouvrante.
2	Il manque un champ.
3	Cette fonction ne peut être appliquée que sur un champ appartenant à une sous-table.
4	Les arguments de la liste doivent tous être du même type.
5	Impossible de déterminer sur quelle table appliquer cette fonction
6	Cette fonction ne peut être exécutée que sur un champ de type sous-table.
7	Il manque un argument de type numérique.
8	Il manque un argument de type alphanumérique.
9	Il manque le résultat d'une condition.
10	Cette fonction ne peut être appliquée à ce type de données.
11	Cette fonction ne peut être appliquée entre deux conditions.
12	Cette fonction ne peut être appliquée entre deux arguments numériques.
13	Cette fonction ne peut être appliquée entre deux arguments alphanumériques.
14	Cette fonction ne peut être appliquée entre deux arguments de type date.
15	Les arguments de cette opération ne sont pas compatibles.
16	Ce champ ne possède pas de lien.
17	Il manque une table.
18	Les types sont incompatibles.
19	Le champ n'est pas indexé.
20	Il manque le signe égal (=).
21	Cette méthode n'existe pas.
22	Les champs doivent appartenir à la même table (ou à la même sous-table) pour un tri ou un graphe.
23	Il manque le signe inférieur (<) ou supérieur (>).
24	Il manque un point-virgule (;).
25	Il y a trop de champs pour le tri.
26	Le champ ne doit pas être de type image, texte, BLOB ou sous-table.
27	Le nom du champ doit être préfixé par le nom de la table auquel il appartient.
28	Le champ doit être du type numérique.
29	La valeur doit être égale à 1 ou 0.
30	Il manque une variable.
31	Aucune barre de menus ne porte ce numéro.
32	Il manque une date.
33	Méthode ou fonction non implémentée
34	Les fichiers comptables ne sont pas ouverts.
35	La table et l'ensemble ne sont pas associés.
36	Nom de table incorrect
37	Il manque le signe d'affectation (:=).
38	Ceci est une fonction et non une méthode.
39	Cet ensemble n'existe pas.
40	Ceci est une méthode et non une fonction.
41	Il manque une variable ou un sous-champ.
42	L'enregistrement ne peut pas être dépilé.
43	La fonction est introuvable.
44	La méthode est introuvable.
45	Il manque une variable ou un champ.
46	Il manque un argument de type alphanumérique ou numérique.
47	Le champ doit être de type alphanumérique.
48	Erreur de syntaxe
49	Impossible d'utiliser cet opérateur ici
50	Ces opérateurs ne peuvent pas être utilisés conjointement.
51	Ce module n'est pas implémenté.
52	Il manque un argument de type tableau.
53	L'indice du tableau est en dehors des limites.
54	Les arguments sont incompatibles.
55	Il manque un argument de type booléen.
56	Il manque un champ, une variable ou une table.
57	Il manque un opérateur.
58	Il manque une parenthèse fermante.
59	Type d'argument inattendu
60	Impossible de passer un paramètre ou une variable locale à une commande EXECUTER sur une base compilée

61	Impossible de modifier le type d'un tableau dans une base compilée
62	Impossible d'appliquer cette commande à une sous-table
63	Le champ n'est pas indexé.
64	Il manque un champ ou une variable de type image.
65	La valeur doit comporter 4 caractères.
66	La valeur doit être composée d'au plus 3 caractères.
67	Cette commande ne peut pas être exécutée sur 4D Server.
68	Il manque une liste.
69	Il manque une référence d'une fenêtre externe.
70	Cette fonction ne peut être appliquée entre deux arguments de type image.
71	La commande **FIXER TAQUET IMPRESSION** peut uniquement être appelée dans l'en-tête d'un formulaire en cours d'impression.
72	Il manque un tableau pointeur.
73	Il manque un tableau numérique.
74	La taille des tableaux ne correspond pas.
75	Pas de pointeur sur les tableaux locaux.
76	Type de tableau erroné.
77	Nom de variable erroné.
78	Paramètre de tri invalide.
79	Cette commande ne peut pas être exécutée pendant le dessin d'une liste.
80	Trop de lignes de recherche.
81	Le formulaire n'a pas été trouvé.

Astuces

Certains codes d'erreurs signalent des erreurs de syntaxe dûes à des fautes de frappe. Par exemple, vous obtenez l'erreur *37* ("Il manque le signe d'affectation (:=).") si vous exécutez l'expression *v=0* alors que vous vouliez écrire *v:=0*. Dans ce cas, vous éliminez l'erreur en corrigeant votre code dans l'éditeur de méthodes.

Certains codes d'erreurs signalent de simples erreurs de programmation. Par exemple, vous obtenez l'erreur *5* ("Impossible de déterminer sur quelle table appliquer cette fonction.") si vous avez exécuté une commande telle que **AJOUTER ENREGISTREMENT** sans indiquer de nom de table dans le paramètre correspondant, et vous n'avez pas défini de table par défaut à l'aide de la commande **TABLE PAR DEFAUT**. Dans ce cas, vous corrigez l'erreur en définissant une table par défaut ou en passant un nom de table dans le paramètre correspondant.

Certains codes d'erreurs signalent des erreurs liées à la structure de la base. Par exemple, vous obtenez l'erreur *16* ("Ce champ ne possède pas de lien.") si vous appliquez la commande **CHARGER SUR LIEN** à un champ qui n'est pas lié à un autre champ. Dans ce cas, vous éliminez l'erreur en modifiant votre code ou en créant un lien à partir du champ.

Certaines erreurs qui surviennent ne stoppent pas toujours l'exécution de votre code au "bon" endroit. Par exemple, si dans une sous-routine vous recevez l'erreur *53* ("L'indice du tableau est en dehors des limites.") sur la ligne *vpChamp:=Champ($1;$2)*, l'erreur est dûe à des numéros incorrects de table ou de champ passés à la sous-routine en tant que paramètres. Donc, l'erreur se trouve dans la méthode appelante et non à l'endroit où l'erreur est détectée. Dans ce cas, tracez votre code dans la fenêtre de débogage et recherchez la ligne qui contient l'erreur, puis corrigez-la dans l'éditeur de méthodes.

Erreurs de la base de données (-10602 -> 4004)

Le tableau suivant liste les codes d'erreurs générées par le moteur de base de données de 4D. Ces erreurs de bas niveau peuvent se produire lors d'opérations liées au moteur telles que des interruptions utilisateur, des erreurs de privilèges ou des objets endommagés.

Code	Description
-10602	Impossible d'exécuter la commande car il n'y a aucune tâche d'impression ouverte.
-10601	La commande **OBJET DUPLIQUER** ne fonctionne pas dans un contexte d'impression.
-10600	Impossible de lire ce BLOB. Il est peut-être endommagé.
-10532	Impossible d'ouvrir la méthode d'appel sur erreur 'nomMéthode'.
-10526	Impossible de créer la base {database_name}.
-10525	Impossible de supprimer base existante {database_name}.
-10524	Impossible d'ouvrir la base {database_name}: le fichier de données est introuvable à sa dernière position connue: '{datafile_path}'.
-10523	Impossible d'ouvrir de la base {database_name}: fichier de données non défini.
-10522	Impossible de charger le composant {database_name} car il n'est pas en mode Unicode.
-10521	Impossible d'ouvrir la base {database_name} car elle n'est pas en mode Unicode.
-10520	Utilisateur non autorisé : Seul l'Administrateur ou le Super_Utilisateur peut exécuter cette commande.
-10519	Url non conforme: {assertion}
-10518	Fausse assertion : {assertion}
-10517	Echec lors de la synchronisation du dossier {folder_name}.
-10516	Le serveur est en cours de maintenance, veuillez vous reconnecter plus tard.
-10515	Votre tentative de connexion à 4D Server a été refusée
-10514	Le nombre maximum d'utilisateurs simultanés pour votre licence a été atteint
-10513	Impossible d'appeler la commande {command_name} d'un 4D Developer distant
-10512	L'encodage n'est pas supporté
-10511	La commande "{command_name}" ne peut pas être appelée depuis un composant.
-10510	Impossible de charger le composant "{component_name}".
-10509	Impossible d'ouvrir la base de données "{database_name}".
-10508	Méthode projet introuvable
-10507	Cette version n'autorise pas l'ouverture d'une base compilée
-10506	Limite de la version Standard Edition
-10505	Client et serveur ont des numéros de version incompatibles
-10504	Numéro de page d'index non valide
-10503	Numéro d'enregistrement non valide
-10502	Structure d'enregistrement non valide
-10501	Structure d'index non valide.
-10500	Adresse de donnée non valide.
-9999	Disque saturé. Impossible de sauvegarder l'enregistrement.
-9998	La clé d'index existe déjà.
-9997	Le nombre maximum d'enregistrements est atteint.
-9996	La pile est pleine (trop d'appels récursifs ou en cascade).

-9995	Limite de la version de démonstration.
-9994	Communication série interrompue par l'utilisateur. L'utilisateur a appuyé sur les touches Ctrl+Alt+Maj (Windows) ou Commande+Option+Maj (Mac OS).
-9993	Barre de menus endommagée (la base de données doit être réparée).
-9992	Ce mot de passe existe déjà.
-9991	Vous n'avez pas l'autorisation d'accès.
-9990	Dépassement du délai en réception.
-9989	Structure de la base invalide (la base de données doit être réparée).
-9988	Impossible de charger ce formulaire.
-9987	D'autres enregistrements sont liés à celui-ci.
-9986	En attente du déverrouillage d'un enregistrement par le process n°
-9985	Détection d'une boucle lors de la suppression
-9984	Détection d'une clé déjà existante lors d'une transaction.
-9983	Attention ! Vous avez installé deux fois le même package de routines externes.
-9982	Enregistrement non chargé car hors sélection pour le poste client.
-9981	Table de définition de nom/numéro de champ envoyée par le poste client incorrecte.
-9980	Création de table impossible car la structure est verrouillée.
-9979	Impossible d'afficher les informations utilisateur.
-9978	Mot de passe incorrect.
-9977	Cette sélection n’existe pas.
-9976	Cette commande ne peut être exécutée car la base est en cours de sauvegarde.
-9975	Page d'index de transaction non chargeable.
-9974	Cet enregistrement vient d'être détruit.
-9973	Mauvaise ressource TRIC.
-9972	Le numéro de la table est en-dehors de l'intervalle défini par le poste client.
-9971	Le numéro du champ est en-dehors de l'intervalle défini par le poste client.
-9970	Le champ n'est pas indexé.
-9969	Type de champ incorrect.
-9968	Numéro d'enregistrement hors sélection.
-9967	L'enregistrement ne peut pas être modifié car il ne peut pas être chargé.
-9966	Les types sont incompatibles.
-9965	Table de définition de recherche incorrecte envoyée par un poste client.
-9964	Table de définition de tri incorrecte envoyée par un poste client.
-9963	Numéro d'enregistrement non valide.
-9962	Pas de sauvegarde possible car le serveur quitte.
-9961	Le process de sauvegarde n'est pas démarré.
-9960	Aucun plug-in de sauvegarde n'est installé.
-9959	Le process de sauvegarde est déjà démarré.
-9958	Le process ne peut être démarré.
-9957	L'énumération est verrouillée.
-9956	Les versions de 4D Server et 4D Client sont incompatibles.
-9955	QuickTime n'est pas installé.
-9954	Aucun enregistrement courant.
-9953	Il n'y a pas de fichier d'historique.
-9952	Mauvais en-tête du segment principal.
-9951	Ce champ ne possède pas de lien.
-9950	Le numéro d'ordre de ce segment de données n'est pas le bon.
-9949	Erreur de licence ou de privilège.
-9948	Une fenêtre modale est active.
-9947	L'option “Autoriser les connexions 4D Open” n'est pas sélectionnée.
-9946	Impossible d'effacer cette sélection temporaire car elle n'existe pas.
-9945	Erreur 4D Runtime CD-ROM, l'écriture de données est impossible.
-9944	Cet utilisateur n'appartient pas au groupe d'accès par 4D Open.
-9943	Erreur de version de plug-in de connectivité 4D.
-9942	Licence 4D Client incompatible avec cette version de 4D Server.
-9941	Sélecteur EX_GESTALT inconnu.
-9940	L'initialisation de l'extension 4D a échoué.
-9939	Routine externe introuvable.
-9938	L'enregistrement courant a été modifié depuis le trigger.
-9937	Le système de mots de passe est verrouillé par un autre utilisateur.
-9936	Le code de mot de passe externe ne correspond pas à celui de la base
-9935	Le fichier XML n’est pas valide ou n’est pas bien formé.
-9934	Le fichier XML n’est pas bien formé.
-9933	Le fichier XML n’est pas valide.
-9932	La DLL XML n’est pas chargée.
-9931	L’index pour cet attribut est invalide.
-9930	Il n’existe pas d’attribut de ce nom pour cet élément.
-9929	L’index pour cet élément est invalide.
-9928	Le nom de l’élément est inconnu.
-9927	L’élément référencé n’est pas le “root”.

-9926	L’élément référencé est invalide.
-9925	L’élément référencé est nul.
-9924	Le fichier doit être ouvert en lecture seule
-9923	Le nom de l'attribut est invalide
-9922	Il manque le paramètre valeur dans la définition des attributs
-9921	Tentative d'écrire un prologue XML dans un document non vide
-9920	Le type du noeud est incorrect
-9919	Cet encodage n'est pas supporté.
-9918	Le nom de l’élément est incorrect.
-9917	Le type du tableau passé en paramètre est incorrect.
-9916	L’élément n'est pas ouvert.
-9915	La référence du document est incorrecte.
-9914	Erreur interne
-9913	Erreur réseau
-9912	Erreur HTTP
-9911	Erreur de l’analyseur xml
-9910	Erreur Web Service
-9909	Pas de fenêtre disponible pour exécuter le formulaire.
-9855	Valeur incorrecte pour le paramètre numéro 5.
-9854	Valeur incorrecte pour le paramètre numéro 4.
-9853	Valeur incorrecte pour le paramètre numéro 3.
-9852	Valeur incorrecte pour le paramètre numéro 2.
-9851	Valeur incorrecte pour le paramètre numéro 1.
-9850	La zone passée à cette commande externe est incorrecte.
-9803	Un objet sans nom a été trouvé dans le formulaire "{form}".
-9802	Chemin d'objet non unique: {path}. Vérifiez l'application avec le CSM.
-9801	Impossible d'ouvrir la méthode : {path}
-9800	L'un des process a modifié les droits d'accès.
-9799	Erreur lors de l'initialisation de la prévisualisation.
-9778	Une erreur est survenue lors du chargement du dictionnaire
-9777	La langue de la méthode ne correspond pas à celle de l’application: {path}
-9776	Impossible de créer la méthode: {path}
-9775	Impossible d'écrire le code de la méthode: {path}
-9774	Impossible de lire le code de la méthode: {path}
-9773	Impossible d'écrire les commentaires de la méthode: {path}
-9772	Impossible de lire les commentaires de la méthode: {path}
-9771	Impossible d'écrire les propriétés de la méthode: {path}
-9770	Impossible de lire les propriétés de la méthode: {path}
-9769	Propriété d'objet invalide: {path}
-9768	Chemin d'objet invalide: {path}
-9767	Impossible de mettre à jour les méthodes. Un ou plusieurs objets sont verrouillés.
-9766	La méthode est en cours d'édition.
-9765	Un ou plusieurs commentaires sont en cours d'édition.
-9764	Le commentaire est en cours d'édition.
-9763	La commande ne peut pas être executée dans un composant.
-9762	La commande ne peut pas être executée dans une base compilée.
-9761	Type d'objet invalide.
-9760	L'élément XML ne peut être déplacé ou copié à la position indiquée.
-9759	La bibliothèque d’objets n’a pas pu être ouverte.
-9758	Le formulaire utilisateur existe déjà.
-9757	Le formulaire utilisateur n’existe pas.
-9756	Il n’y a pas de fichier de structure utilisateur.
-9755	Le formulaire utilisateur n’a pas de nom.
-9754	Cette commande ne peut pas être appelée depuis une fenêtre de dialogue.
-9753	Le formulaire source n’existe pas.
-9752	Le formulaire utilisateur ne peut pas être créé.
-9751	Le formulaire source n’est pas accessible pour l’utilisateur.
-9750	Le formulaire source n’est pas modifiable.
-1	Point d'entrée non valide utilisé par un plug-in
1001	Pas de colonne à importer dans la table {TableName}
1002	Table d’import incorrecte
1003	Pas de colonne à exporter dans la table {TableName}
1004	Table d’export incorrecte
1006	Impossible de sauvegarder la structure de la base {BaseName}
1006	Interruption générée par l'utilisateur.
1007	Impossible de créer le fichier de données de la base : {BaseName}
1008	Segment de données incorrect dans la base {BaseName}
1009	La mémoire est pleine
1010	Impossible de charger la définition des tables de la base {BaseName}
1011	>Impossible d'ouvrir le fichier de données de la base {BaseName}

1012 Le segment de données est plein dans la base : {BaseName}
1013 Impossible de sauvegarder le segment de données dans la base {BaseName}
1014 Impossible de lire le segment de données de la base {BaseName}
1015 En-tête de base de données incorrect dans la base {BaseName}
1016 Impossible de créer la table dans la base {BaseName}
1017 Impossible de lire la liste des index de la base {BaseName}
1018 Impossible d'écrire la liste des index de la base {BaseName}
1019 Référence de table incorrecte dans la base {BaseName}
1020 Référence de champ incorrecte dans la base {BaseName}
1021 Type d'index invalide dans la base {BaseName}
1022 Nom de champ invalide pour la table {TableName} de la base {BaseName}
1023 Nom de base de données invalide
1024 Impossible d'ouvrir la structure de la base {BaseName}
1025 Impossible de créer la structure de la base {BaseName}
1026 Impossible de charger la 'bit selection' de la base {BaseName}
1027 Impossible de charger l'ensemble de la base {BaseName}
1028 Impossible de modifier l'ensemble de la base {BaseName}
1029 Impossible de sauvegarder l'ensemble de la base {BaseName}
1030 Impossible de sauvegarder le BLOB {BlobNum} dans la table {TableName} de la base {BaseName}
1031 Impossible de charger le BLOB {BlobNum} dans la table {TableName} de la base {BaseName}
1032 Impossible d'allouer le BLOB {BlobNum} dans la table {TableName} de la base {BaseName}
1033 Impossible de charger la table de bits des données de la base {BaseName}
1034 Impossible de sauvegarder la table de bits des données de la base {BaseName}
1035 Impossible de charger la table de bits des données de la base {BaseName}
1036 Impossible de sauvegarder la table de bits des données dans la base {BaseName}
1037 Impossible de fermer le segment de données de la base {BaseName}
1038 Impossible de supprimer le segment de données de la base {BaseName}
1039 Impossible d'ouvrir le segment de données de la base {BaseName}
1040 Impossible de créer le segment de données de la base {BaseName}
1041 Impossible d'allouer l'espace dans le segment de données
1042 Impossible de libérer l'espace dans le segment de données de la base {BaseName}
1043 Le fichier est protégé en écriture
1044 Impossible d'accéder au champ dans l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1045 Le code de définition du champ est manquant dans l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1046 Impossible de sauvegarder l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1047 Impossible de charger l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1048 Impossible d'allouer l'enregistrement de la table {TableName} de la base {BaseName}
1049 Impossible de mettre à jour l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1050 En-tête incorrect pour l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1051 Impossible de sauvegarder la définition de la table {TableName} de la base {BaseName}
1052 Impossible de mettre à jour la définition de la table {TableName} de la base {BaseName}
1053 Le nom du champ existe déjà
1055 Impossible de mettre à jour les valeurs d'index pendant la sauvegarde d'un enregistrement de la table {TableName} de la base {BaseName}
1056 Impossible de mettre à jour les BLOBs pendant la sauvegarde d'un enregistrement de la table {TableName} de la base {BaseName}
1057 Impossible de supprimer les BLOBs pendant la sauvegarde ou la suppression d'un enregistrement de la table {TableName} de la base {BaseName}
1058 Impossible d'ajouter le champ dans la table {TableName} de la base {BaseName}
1059 Impossible d'allouer la table dans la mémoire
1061 Impossible d'allouer l'enregistrement dans la mémoire
1062 Impossible de supprimer complètement la table {TableName} de la base {BaseName}
1063 >Impossible de verrouiller l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1064 Impossible de déverrouiller l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1065 Impossible de supprimer l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1066 L'enregistrement {RecNum} est verrouillé dans la table {TableName} de la base {BaseName}
1067 La recherche séquentielle n'a pas pu se terminer dans la table {TableName} de la base {BaseName}
1068 Impossible de sauvegarder l'en-tête de la table {TableName} de la base {BaseName}
1069 Impossible d'importer les données de la table {TableName} de la base {BaseName}
1070 Impossible de charger l'en-tête de l'index de la base {BaseName}
1071 Impossible de sauvegarder l'en-tête de l'index {IndexName} de la base {BaseName}
1072 Impossible de lire l'adresse de la page d'index {IndexName} de la base {BaseName}
1073 Impossible d'écrire l'adresse de la page d'index {IndexName} de la base {BaseName}
1074 Impossible de supprimer l'index {IndexName} de la base {BaseName}
1075 Impossible de trier l'index {IndexName} de la base {BaseName}
1076 Impossible de charger la page d'index {IndexName} de la base {BaseName}
1077 Impossible de sauvegarder la page d'index {IndexName} de la base {BaseName}
1078 Impossible d'insérer la clé dans l'index {IndexName} de la base {BaseName}
1079 Impossible de supprimer la clé de l'index {IndexName} de la base {BaseName}
1080 Impossible de supprimer complètement l'index {IndexName} de la base {BaseName}
1081 Impossible d'accomplir le balayage de l'index {IndexName} de la base {BaseName}
1082 Impossible d'accomplir le tri de l'index {IndexName} de la base {BaseName}

1083 Impossible d’insérer la clé dans la page d’index {IndexName} de la base {BaseName}
1084 Impossible de supprimer la clé dans la page d’index {IndexName} de la base {BaseName}
1085 Impossible de charger le cluster d’index de la base {BaseName}
1086 Impossible d’ajouter au cluster d’index de la base {BaseName}
1087 Impossible de supprimer dans le cluster d’index de la base {BaseName}
1088 L'index {IndexName} est invalide
1089 Erreur de syntaxe SQL (Obsolète)
1090 Mot-clé SQL non trouvé (Obsolète)
1091 Non implémenté
1092 Impossible d’enregistrer le code
1093 Opération en cours annulée par l’utilisateur
1094 Conflit de transaction
1095 Nom de table invalide dans la base {BaseName}
1096 La table {TableName} est verrouillée dans la base {BaseName}
1097 La base {BaseName} est verrouillée
1098 L’adresse de données est invalide dans la base {BaseName}
1099 L’enregistrement est vide
1100 Champ source incorrect
1101 Champ destination incorrect
1102 Nom de lien invalide
1103 Types de champ incompatibles
1104 Le lien est vide
1105 Impossible de charger la liste des liens à partir de la base {BaseName}
1106 Impossible de sauvegarder la liste des liens dans la base {BaseName}
1107 Requête et verrouillage incomplets, un enregistrement au moins était verrouillé
1108 Enregistrement invalide
1109 Numéro d’enregistrement incorrect
1110 Le lien existe déjà dans la base {BaseName}
1111 L’index existe déjà dans la base {BaseName}
1112 Opérateur de comparaison incorrect
1113 Fin du buffer de données
1114 Numéro de version DB4D incorrect
1115 Clé dupliquée
1116 Le champ obligatoire est Null dans l'enregistrement {RecNum} de la table {TableName}
1117 Impossible d’appliquer l’attribut Obligatoire au champ de la table {TableName}
1118 Impossible d’obtenir un accès exclusif à la table {TableName}
1119 Impossible de vérifier l’intégrité référentielle de la table {TableName}
1120 Intégrité référentielle : il y a encore des clés externes correspondant à la clé primaire de l'enregistrement {RecNum} de la table {TableName}
1121 Impossible de supprimer tous les enregistrements de la sélection de la table {TableName}
1122 Le BLOB est Null
1123 Contexte de base de données incorrect
1124 Référence de lien invalide
1125 Nom d’enregistrement incorrect dans la table {TableName}
1126 Type de champ incorrect
1127 Impossible de charger les propriétés additionnelles
1128 Impossible de sauvegarder les propriétés additionnelles
1129 Le numéro du sous-enregistrement est en dehors des limites
1130 Nom dupliqué pour l'index {IndexName} dans la base {BaseName}
1131 Nom invalide pour l'index {IndexName} dans la base {BaseName}
1132 Valeur de clé invalide pour l'index {IndexName}
1133 Type incorrect pour l'index {IndexName}
1134 Accesseur invalide
1135 L’accesseur est en lecture seulement
1136 Valeur Null non acceptée
1137 THIS est Null
1138 La sélection est Null
1139 La base de données {BaseName} est protégée en écriture
1140 La base de données {BaseName} est en cours de fermeture
1141 Transaction invalide
1142 La limite de tableau est dépassée
1143 Le créateur des valeurs du tableau est manquant
1144 Impossible de construire la sélection de la table {TableName}
1145 Objet actuellement indisponible
1146 Le fichier de données ne correspond pas au fichier de structure
1147 Impossible de lancer le serveur d'écoute réseau
1148 Impossible de lancer le serveur
1149 Pas de serveur d'écoute réseau
1150 Le process est en train de mourir
1151 Balise de requête invalide
1152 Numéro de contexte invalide

1153 Espace disque temporaire insuffisant
1154 L'ensemble de données est Null
1155 Aucune clé primaire ne correspond à la clé externe
1156 Le type du champ {FieldName} de la table {TableName} n'accepte pas l'attribut Unique
1157 Le type du champ {FieldName} de la table {TableName} n'accepte pas l'attribut NEVER NULL (jamais Null)
1158 Impossible de modifier la définition de la clé primaire de la table {TableName}
1159 Le nombre maximal d'enregistrements a été atteint pour la table {TableName}
1160 Le nombre maximal de BLOBs a été atteint pour la table {TableName}
1161 L'indice est en dehors des limites
1162 La requête est invalide
1163 L'enregistrement est Null
1164 L'objet est Null
1165 Propriétaire incorrect pour cet objet
1166 L'objet n'était pas verrouillé
1167 L'objet est verrouillé par un autre contexte
1168 Erreur interne sur une connexion distante
1169 Numéro de table invalide
1170 Numéro de champ invalide
1171 Numéro de base de données invalide
1172 Paramètre invalide
1173 L'écriture du cache est incomplète
1174 L'écriture des données est incomplète
1175 L'écriture de la structure est incomplète
1176 Le fichier d'historique est invalide pour la base {BaseName}
1177 Le fichier d'historique est introuvable pour la base {BaseName}
1178 La dernière opération du fichier d'historique ne correspond pas à la base {BaseName}
1179 Le fichier d'historique ne correspond pas à la base {BaseName}
1180 Les données de la table ont été supprimées
1181 Les clés ne sont pas uniques dans l'index {IndexName}
1182 Impossible de créer un fichier d'historique pour la base {DataBase}
1183 Impossible d'écrire dans le fichier d'historique de la base {DataBase}
1184 Impossible de déposer la table {TableName} dans la base {DataBase}
1185 La base de données distante ne peut pas être ouverte
1186 Le fichier d'historique ne peut pas être intégré à la base {DataBase}
1187 L'opération interne sur l'ensemble est incomplète
1188 Le tableau ne peut pas être sauvegardé
1189 Le tableau ne peut pas être chargé
1190 L'en-tête du numéro de séquence ne peut pas être chargé
1191 Impossible de sélectionner l'enregistrement
1192 L'enregistrement ne peut pas être créé
1193 Impossible d'accomplir sélection vers tableau de la table {TableName} de la base {BaseName}
1194 Impossible d'accomplir tableau vers sélection de la table {TableName} de la base {BaseName}
1195 Impossible d'accomplir le tri séquentiel
1196 La sélection ne peut pas être verrouillée
1197 La clé d'index ne peut pas être chargée
1198 La clé d'index ne peut pas être sauvegardée
1199 La clé d'index ne peut pas être construite
1200 La requête ne peut pas se terminer
1201 La requête ne peut pas être analysée
1202 La formule n'a pas pu être effectuée sur cette colonne
1203 La requête n'a pas pu se terminer
1204 La requête n'a pas pu être analysée
1205 Impossible d'obtenir toutes les valeurs distinctes de la table {TableName} de la base {BaseName}
1206 Impossible de construire le tableau de valeurs de la table {TableName} de la base {BaseName}
1207 La sélection ne peut pas être chargée
1208 Impossible d'envoyer les données
1209 Impossible de recevoir des données
1210 Impossible d'envoyer une requête
1211 Impossible de créer une connexion
1212 L'index {IndexName} ne peut pas être créé rapidement dans la base {BaseName}
1213 Impossible de construire les clés d'index distinctes
1214 La sélection ne peut pas être triée dans la table {TableName} de la base {BaseName}
1215 La table d'adresse ne peut pas être chargée
1216 La table d'adresse ne peut pas être modifiée
1217 mpossible d'allouer une nouvelle entrée à la table d'adresse
1218 Impossible d'allouer une entrée vide de la table d'adresse
1219 L'enregistrement temporaire de transaction ne peut pas être sauvegardé
1220 Le blob temporaire de transaction ne peut pas être sauvegardé
1221 L'enregistrement temporaire de transaction ne peut pas être chargé
1222 Le blob temporaire de transaction ne peut pas être chargé

1223	La transaction ne peut pas être démarrée
1224	La transaction ne peut pas être validée
1225	Impossible de lire la propriété additionnelle
1226	Impossible d'écrire la propriété additionnelle
1227	Le nom de la table est déjà utilisé
1228	Impossible de lire la liste des clés NULL de l'index {IndexName}
1229	Impossible de modifier la liste des clés NULL de l'index {IndexName}
1230	Clé invalide pour l'index {IndexName}
1231	Impossible d'écrire le fichier d'historique
1232	Le contexte est NULL
1233	La base {BaseName} ne peut pas être verrouillée
1234	Référence de champ incorrecte dans l'enregistrement# {RecNum} de la table : {TableName} de la base : {BaseName}
1235	Référence de champ incorrecte dans la table : {TableName} de la base : {BaseName}
1236	Impossible de lire les données du fichier de transaction temporaire
1237	Le produit cartésien échoué
1238	Impossible de fusionner sélections
1239	Le format de la base {BaseName} ne peut pas être mis à jour en mode lecture seulement
1240	En-tête incorrect
1241	CheckSum incorrect
1243	Impossible de charger la table des données de la base : {BaseName}
1244	La liste de contraintes des clés étrangères n'est pas vide pour la table : {TableName}dans la base : {BaseName}
1245	L'entrée d'adresse n'est pas vide
1246	Impossible de pré-allouer l'adresse
1247	Impossible de mettre à jour le nouvel enregistrement dans la table {TableName} de la base {BaseName}
1248	Impossible de sauvegarder le nouvel enregistrement dans la table {TableName} de la base {BaseName}
1249	Impossible de sauvegarder le sous-enregistrement
1250	Impossible de sauvegarder l'enregistrement
1251	Impossible de verrouiller la définition de l'objet de structure
1252	Impossible de déverrouiller la définition de l'objet de structure
1253	Numéro de lien invalide
1254	Référence circulaire de la table d'adresse des enregistrements de la table {TableName} de la base {BaseName}
1255	Référence circulaire dans la table d'adresse des blobs de la table {TableName} de la base {BaseName}
1256	Nom de schéma dupliqué dans la base {BaseName}
1257	Le schéma ne peut être sauvegardé dans la base {BaseName}
1258	Le schéma ne peut être supprimé dans la base {BaseName}
1259	Le schéma ne peut être renommé dans la base {BaseName}
1260	Le fichier d'historique est trop récent pour la base {BaseName}
1261	Le fichier d'historique est trop ancien pour la base {BaseName}
1262	Certaines DataTables n'ont pas de définitions de tables correspondantes dans la base de données {BaseName}
1263	Le marqueur fourni ne correspond pas à celui de l'enregistrement# {RecNum} de la table {TableName}
1264	Une clé primaire est nécessaire et est manquante dans la table {TableName}
1265	Champ invalide
1266	Type de champ inconnu (cette version de 4D est peut-être trop ancienne)
1267	Dépassement de la capacité de la pile
1268	La table ne peut pas être régénérée pour la base {BaseName}
1269	La table ne peut pas être trouvée
1270	La structure manquante ne peut pas être recréée
1271	Gestionnaire de requête REST invalide
1272	Certains enregistrements de la table {tableName} sont encore verrouillés dans la base {BaseName}
1273	Impossible de supprimer la table {TableName} dans la base {BaseName}
1274	Le fichier d'historique de la base {BaseName} n'est pas disponible. Pour des raisons de sécurité des données, les opérations d'écriture sont interrompues. Veuillez contacter votre administrateur dès que possible.
1275	"(" est manquante au début de l'expression Javascript dans cette requête.
1276	Datastream a un en-tête invalide.
1277	L'intégration du fichier d'historique de la base a échoué à l'entrée# {p1}.
1278	Code Javascript non autorisé dans cette requête.
1300	Type de sélection invalide
1301	Le tableau est trop grand
1302	La taille du tableau ne correspond pas
1303	ID de la sélection invalide
1304	Partie de la sélection invalide
4001	Numéro de table non valide utilisé par un plug-in
4002	Numéro d'enregistrement non valide utilisé par un plug-in
4003	Numéro de champ invalide utilisé par un plug-in
4004	Un plug-in a requis l'enregistrement courant d'une table alors qu'il n'y en a pas

Notes

1. Bien que certaines de ces erreurs signalent des problèmes sérieux — par exemple *(-10502), Structure d'enregistrement non valide* — la plupart sont relativement courantes et peuvent peuvent être traitées par une méthode projet **APPELER SUR ERREUR**. Par exemple, vous intercepterez fréquemment l'erreur *–9998, La clé d'index existe déjà* si votre application laisse la possibilité de créer des valeurs identiques pour une table qui contient un champ indexé ayant la propriété Unique.

2. Certaines de ces erreurs ne se produisent jamais au niveau du langage de 4D. Elles ne surviennent et ne peuvent être traitées qu'à un bas niveau par des routines du moteur de la base ou pendant l'utilisation, par exemple, de 4D Open.
3. L'erreur *-10503, Numéro d'enregistrement non valide* signifie généralement que votre code (par exemple la commande **ALLER A ENREGISTREMENT**) tente d'accéder à un enregistrement qui n'existe pas ou plus. Dans certains cas, plus rares, cette erreur peut signifier que la base doit être réparée.
4. L'erreur *-9999 Disque saturé. Impossible de sauvegarder l'enregistrement* se produit lorsque le fichier de données de votre base est plein ou placé sur un volume plein. Cette erreur peut également être générée si le fichier de données est verrouillé ou stocké sur un volume verrouillé.

Erreurs du moteur SQL (1001 -> 3018)

Le moteur SQL de 4D retourne des erreurs spécifiques, listées ci-dessous. Ces erreurs peuvent être interceptées à l'aide d'une méthode gestion d'erreurs installée par la commande **APPELER SUR ERREUR** et analysées via la commande **LIRE PILE DERNIERE ERREUR**.

Erreur génériques

1001	INVALID ARGUMENT
1002	INVALID INTERNAL STATE
1003	NOT RUNNING
1004	Accès refusé
1005	FAILED TO LOCK SYNCHRONIZATION PRIMITIVE
1006	FAILED TO UNLOCK SYNCHRONIZATION PRIMITIVE
1007	SQL SERVER IS NOT AVAILABLE
1008	COMPONENT BRIDGE IS NOT AVAILABLE
1009	REMOTE SQL SERVER IS NOT AVAILABLE
1010	L'exécution a été interrompue par l'utilisateur.

Erreurs sémantiques

1101 La table '{key1}' n'existe pas dans la base de données.
1102 La colonne '{key1}' n'existe pas.
1103 La table '{key1}' n'est pas déclarée dans la clause FROM.
1104 La référence vers le nom de la colonne '{key1}' est ambigüe.
1105 L'alias '{key1}' de la table est identique au nom de la table.
1106 Alias de tables dupliqués
1107 Table dupliquée dans la clause FROM - '{key1}'.
1108 L'opération {key1} {key2} {key3} n'est pas de type valide.
1109 Index invalide dans la clause ORDER BY - {key1}.
1110 La fonction {key1} attend un paramètre, pas {key2}.
1111 Le paramètre {key1} de type {key2} dans l'appel de la fonction {key3} n'est pas convertible implicitement en {key4}.
1112 Fonction inconnue - {key1}.
1113 Division par zéro.
1114 Le tri par article indexé dans la liste SELECT n'est pas autorisé - article {key1} dans la clause ORDER BY.
1115 DISTINCT NOT ALLOWED
1116 Les fonctions statistiques imbriquées ne sont pas autorisées dans la fonction statistique {key1}.
1117 La sous requête ne doit pas contenir de fonctions de colonnes dans le contexte de sa super-requête.
1118 Impossible de mixer des opérations de colonne et scalaires.
1119 Index invalide dans la clause GROUP BY - {key1}.
1120 L'index GROUP BY n'est pas autorisé.
1121 GROUP BY n'est pas autorisé avec 'SELECT * FROM ...'.
1122 HAVING n'est pas une expression statistique
1123 La colonne '{key1}' n'est pas une colonne groupée et ne peut pas être utilisée dans la clause ORDER BY.
1124 Impossible de mixer les types {key1} et {key2} dans le prédicat IN.
1125 La séquence d'échappement '{key1}' dans le prédicat LIKE est trop longue. Ce doit être exactement un caractère.
1126 Caractère d'échappement erroné - '{key1}'.
1127 Séquence d'échappement inconnue - '{key1}'.
1128 Les références des colonnes de plus d'une requête dans la fonction statistique {key1} n'est pas autorisé.
1129 L'élément scalaire dans la liste SELECT n'est pas autorisé quand la clause GROUP BY est présente.
1130 Les sous requêtes produisent plus d'une colonne.
1131 La sous requête doit retourner une ligne au maximum mais ici elle renvoie {key1}.
1132 Le nombre de valeurs dans INSERT {key1} ne correspond pas au nombre de colonnes {key2}.
1133 Référence de colonne dupliquée dans la liste INSERT - '{key1}'.
1134 La colonne '{key1}' n'autorise pas les valeurs NULL.
1135 Référence de colonne dupliquée dans la liste UPDATE - '{key1}'.
1136 Table '{key1}' déjà existante.
1137 Colonne '{key1}' dupliquée dans la commande CREATE TABLE.
1138 DUPLICATE COLUMN IN COLUMN LIST
1139 Une seule clé primaire est autorisée.
1140 Nom de clé externe ambigu - '{key1}'.
1141 Le nombre de colonnes {key1} dans la table enfant ne correspond pas au nombre de colonnes {key2} dans la table parent de la clé externe.
1142 Incompatibilité de types de colonne dans la définition de la clé externe. Impossible de lier {key1} dans la table enfant à {key2} dans la table parent.
1143 Impossible de trouver la colonne correspondante dans la table parent pour la colonne '{key1}' dans la table enfant.
1144 Les contraintes UPDATE et DELETE doivent être les mêmes.
1145 FOREIGN KEY DOES NOT EXIST
1146 Valeur invalide pour LIMIT dans la commande SELECT - {key1}.
1147 Valeur invalide pour OFFSET dans la commande SELECT - {key1}.
1148 La clé primaire n'existe pas dans la table '{key1}'.
1149 FAILED TO CREATE FOREIGN KEY
1150 La colonne '{key1}' ne fait pas partie de la clé primaire.
1151 FIELD IS NOT UPDATEABLE
1153 Longueur de type de données incorrecte '{key1}'.
1156 L'auto-increment n'est pas autorisée pour la colonne '{key1}' de type {key2}.
1159 Impossible de supprimer le schéma système.
1160 CHARACTER ENCODING NOT ALLOWED.

Erreurs d'implémentations

1203	FUNCTIONALITY IS NOT IMPLEMENTED
1204	Echec de la création de l'enregistrement {key1}.
1205	Echec de la mise à jour du champ '{key1}'.
1206	Echec de la suppression de l'enregistrement '{key1}'.
1207	NO MORE JOIN SEEDS POSSIBLE
1208	FAILED TO CREATE TABLE
1209	FAILED TO DROP TABLE
1210	CANT BUILD BTREE FOR ZERO RECORDS
1211	COMMAND COUNT GREATER THAN ALLOWED
1212	FAILED TO CREATE DATABASE
1213	FAILED TO DROP COLUMN
1214	VALUE IS OUT OF BOUNDS
1215	FAILED TO STOP SQL_SERVER
1216	FAILED TO LOCALIZE
1217	Impossible de verrouiller la table pour la lecture.
1218	FAILED TO LOCK TABLE FOR WRITING
1219	TABLE STRUCTURE STAMP CHANGED
1220	FAILED TO LOAD RECORD
1221	FAILED TO LOCK RECORD FOR WRITING
1222	FAILED TO PUT SQL LOCK ON A TABLE
1223	FAILED TO RETAIN COOPERATIVE TASK
1224	FAILED TO LOAD INFILE

Erreurs d'analyse

1301	PARSING FAILED

Erreurs d'accès au langage runtime

1401	COMMAND NOT SPECIFIED
1402	ALREADY LOGGED IN
1403	SESSION DOES NOT EXIST
1404	UNKNOWN BIND ENTITY
1405	INCOMPATIBLE BIND ENTITIES
1406	REQUEST RESULT NOT AVAILABLE
1407	BINDING LOAD FAILED
1408	COULD NOT RECOVER FROM PREVIOUS ERRORS
1409	NO OPEN STATEMENT
1410	RESULT EOF
1411	BOUND VALUE IS NULL
1412	STATEMENT ALREADY OPENED
1413	FAILED TO GET PARAMETER VALUE
1414	INCOMPATIBLE PARAMETER ENTITIES
1415	Le paramètre de la requête n'est pas autorisé ou est manquant.
1416	COLUMN REFERENCE PARAMETERS FROM DIFFERENT TABLES
1417	EMPTY STATEMENT
1418	FAILED TO UPDATE VARIABLE
1419	FAILED TO GET TABLE REFERENCE
1420	FAILED TO GET TABLE CONTEXT
1421	COLUMNS NOT ALLOWED
1422	INVALID COMMAND COUNT
1423	INTO CLAUSE NOT ALLOWED
1424	EXECUTE IMMEDIATE NOT ALLOWED
1425	ARRAY NOT ALLOWED IN EXECUTE IMMEDIATE
1426	COLUMN NOT ALLOWED IN EXECUTE IMMEDIATE
1427	NESTED BEGIN END SQL NOT ALLOWED
1428	RESULT IS NOT A SELECTION
1429	INTO ITEM IS NOT A VARIABLE
1430	VARIABLE WAS NOT FOUND
1431	PTR OF PTR NOT ALLOWED
1432	POINTER OF UNKNOWN TYPE

Erreurs d'analyse de date

1501	SEPARATOR_EXPECTED
1502	FAILED TO PARSE DAY OF MONTH
1503	FAILED TO PARSE MONTH
1504	FAILED TO PARSE YEAR
1505	FAILED TO PARSE HOUR
1506	FAILED TO PARSE MINUTE
1507	FAILED TO PARSE SECOND
1508	FAILED TO PARSE MILLISECOND
1509	INVALID AM PM USAGE
1510	FAILED TO PARSE TIME ZONE
1511	UNEXPECTED CHARACTER
1512	Echec de l'analyse du timestamp.
1513	Echec de l'analyse de la durée.
1551	FAILED TO PARSE DATE FORMAT

Erreurs lexer

1601	NULL INPUT STRING
1602	NON TERMINATED STRING
1603	NON TERMINATED COMMENT
1604	INVALID NUMBER
1605	UNKNOWN START OF TOKEN
1606	NON TERMINATED NAME
1607	NO VALID TOKENS

Erreurs de validation - Erreurs de statut suivant les erreurs directes

1701	Echec de la validation de la table '{key1}'.
1702	Echec de la validation de la clause FROM.
1703	Echec de la validation de la clause GROUP BY.
1704	Echec de la validation de la liste SELECT.
1705	Echec de la validation de la clause WHERE.
1706	Echec de la validation de la clause ORDER BY.
1707	Echec de la validation de la clause HAVING.
1708	Echec de la validation du prédicat COMPARISON.
1709	Echec de la validation du prédicat BETWEEN.
1710	Echec de la validation du prédicat IN.
1712	Echec de la validation du prédicat ALL ANY.
1713	Echec de la validation du prédicat EXISTS.
1714	Echec de la validation du prédicat NULL.
1715	Echec de la validation de la sous-requête.
1716	Echec de la validation de l'article SELECT {key1}.
1717	Echec de la validation de la colonne '{key1}'.
1718	Echec de la validation de la fonction '{key1}'.
1719	Echec de la validation de l'expression CASE.
1720	Echec de la validation du paramètre de la commande.
1721	Echec de la validation du paramètre '{key1}' de la fonction.
1722	Echec de la validation de l'article '{key1}' de la liste INSERT.
1723	Echec de la validation de l'article '{key1}' de la liste UPDATE.
1724	Echec de la validation de la liste des colonnes.
1725	Echec de la validation de la clé étrangère.
1726	Echec de la validation de la commande SELECT.
1727	Echec de la validation de la commande INSERT.
1728	Echec de la validation de la commande DELETE.
1729	Echec de la validation de la commande UPDATE.
1730	Echec de la validation de la commande CREATE TABLE.
1731	Echec de la validation de la commande DROP TABLE.
1732	Echec de la validation de la commande ALTER TABLE.
1733	Echec de la validation de la commande CREATE INDEX.
1734	Echec de la validation de la commande LOCK TABLE.
1735	Echec du calcul du modèle du prédicat LIKE.

Erreurs d'exécution - Erreurs de statut suivant les erreurs directes

1801	Echec de l'exécution de la commande SELECT.
1802	Echec de l'exécution de la commande INSERT.
1803	Echec de l'exécution de la commande DELETE.
1804	Echec de l'exécution de la commande UPDATE.
1805	Echec de l'exécution de la commande CREATE TABLE.
1806	Echec de l'exécution de la commande DROP TABLE.
1807	Echec de l'exécution de la commande CREATE DATABASE.
1808	Echec de l'exécution de la commande ALTER TABLE.
1809	Echec de l'exécution de la commande CREATE INDEX.
1810	Echec de l'exécution de la commande DROP INDEX.
1811	Echec de l'exécution de la commande LOCK TABLE.
1812	Echec de l'exécution de la commande TRANSACTION.
1813	Echec de l'exécution de la clause WHERE.
1814	Echec de l'exécution de la clause GROUP BY.
1815	Echec de l'exécution de la clause HAVING.
1816	Echec de l'agrégation.
1817	Echec de l'exécution de DISTINCT.
1818	Echec de l'exécution de la clause ORDER BY.
1819	Echec de la construction de la requête DB4D.
1820	Echec de calcul du prédicat de la comparaison.
1821	Echec de l'exécution de la sous requête.
1822	Echec du calcul du prédicat de BETWEEN.
1823	Echec du calcul du prédicat de IN.
1824	Echec du calcul du prédicat de ALL/ANY.
1825	Echec du calcul du prédicat de LIKE.
1826	Echec du calcul du prédicat de EXISTS.
1827	Echec du calcul du prédicat de IS NULL.
1828	Echec de l'opération arithmétique.
1829	Echec du calcul l'appel de fonction '{key1}'.
1830	Echec du calcul de l'expression Au cas ou.
1831	Echec du calcul du paramètre de la fonction '{key1}'.
1832	Echec du calcul de l'appel de la fonction 4D.
1833	Echec du tri pendant l'exécution de la clause ORDER BY.
1834	Echec du calcul du hash de l'enregistrement.
1835	Echec de la comparaison des enregistrements.
1836	Echec du calcul de la valeur {key1} de INSERT.
1837	SQL DB4D QUERY FAILED
1838	FAILED TO EXECUTE ALTER SCHEMA COMMAND
1839	FAILED TO EXECUTE GRANT COMMAND

Erreurs de cache

2000	CACHEABLE NOT INITIALIZED
2001	VALUE ALREADY CACHED
2002	CACHED VALUE NOT FOUND
2003	CACHE IS FULL
2004	CACHING IS NOT POSSIBLE

Erreurs de protocole

3000	HEADER NOT FOUND
3001	UNKNOWN COMMAND
3002	ALREADY LOGGED IN
3003	NOT LOGGED IN
3004	UNKNOWN OUTPUT MODE
3005	INVALID STATEMENT ID
3006	UNKNOWN DATA TYPE
3007	STILL LOGGED IN
3008	SOCKET READ ERROR
3009	SOCKET WRITE ERROR
3010	BASE64 DECODING ERROR
3011	SESSION TIMEOUT
3012	FETCH TIMESTAMP ALREADY EXISTS
3013	BASE64 ENCODING ERROR
3014	INVALID HEADER TERMINATOR
3015	INVALID SESSION TICKET
3016	HEADER TOO LONG
3017	INVALID AGENT SIGNATURE
3018	UNEXPECTED HEADER VALUE

Erreurs réseau (-10051 -> -10001)

Le tableau suivant liste les erreurs qui peuvent se produire dans le cadre des connexions réseau.

Code	Description
-10051	Valeur incorrecte pour l'option du composant réseau.
-10050	L'option du composant réseau est inconnue.
-10033	Taille des données incorrecte pendant la lecture.
-10031	Désynchronisation pendant la lecture.
-10030	Désynchronisation pendant l'écriture.
-10021	Aucun serveur trouvé.
-10020	Aucun serveur sélectionné.
-10003	Paramètres de connexion incorrects.
-10002	Interruption de la connexion pour ce process ou bien la connexion n'a pu être établie.
-10001	Interruption de la connexion à la base en cours.

Erreurs du gestionnaire de sauvegarde (1401 -> 1421)

Le tableau suivant liste les codes d'erreurs spécifiques générés par le module de sauvegarde et de restitution de 4D.
Ces erreurs peuvent être interceptées par une méthode installée via la commande **APPELER SUR ERREUR**.

Code	Description
1401	Le nombre maximal de tentatives de sauvegarde est atteint, la sauvegarde automatique est temporairement désactivée.
1403	Cette base travaille sans fichier d'historique.
1404	Une transaction est ouverte dans ce process. La sauvegarde ne peut pas démarrer.
1405	Le délai d'attente maximum de fin de transaction dans un process concurrent est atteint. L'opération ne peut être exécutée.
1406	La sauvegarde a été annulée par l'utilisateur.
1407	Le dossier de destination est invalide.
1408	Erreur pendant la sauvegarde du fichier d'historique.
1409	Erreur pendant la sauvegarde.
1410	Le fichier de sauvegarde est introuvable, impossible de le vérifier.
1411	Erreur pendant la vérification du fichier de sauvegarde.
1412	Le fichier de sauvegarde d'historique est introuvable, impossible de le vérifier.
1413	Erreur pendant la vérification du fichier de sauvegarde d'historique.
1414	Cette commande ne peut être exécutée que sur 4D Server.
1415	Impossible de sauvegarder l'historique, une opération critique est en cours.
1416	Ce fichier d'historique ne correspond pas à la base ouverte.
1417	Une opération d'intégration du fichier d'historique est en cours. La sauvegarde ne peut pas démarrer.
1420	Impossible d'intégrer l'historique car un enregistrement au moins est verrouillé dans la base.
1421	Cette commande ne peut pas être utilisée en environnement client/serveur.

- Les erreurs 1408 et 1409 proviennent généralement d'une erreur de lecture des fichiers à sauvegarder ou d'une erreur d'écriture avec les fichiers en cours de sauvegarde.
- Les erreurs 1411 et 1413 se produisent lors de la vérification des archives.
  Lorsque ces erreurs surviennent, il peut être judicieux de vérifier dans un premier temps la place restante sur le disque et les privilèges d'accès en lecture écriture.

Erreurs de la mise à jour d'une application cliente (-10650 -> -10655)

Le tableau suivant liste les codes et les messages des erreurs qui peuvent survenir lors de la mise à jour d'une application cliente :

Code	Description
-10650	Impossible de télécharger la mise à jour de l'application cliente
-10651	Impossible de décompresser la mise à jour de l'application cliente
-10652	Information de mise à jour non disponible ({path})
-10653	Echec du téléchargement de la mise à jour '{path}'
-10654	Information de mise à jour non valide
-10655	La mise à jour de l'application cliente n'est pas disponible

Erreurs de l'outil de mise à jour (-10603 -> -10613)

Le tableau suivant liste les codes et les messages des erreurs qui peuvent survenir lors de l'utilisation de l'outil de mise à jour :

Code	Description
-10603	Impossible d'installer l'outil de mise à jour
-10604	Impossible d'identifier le paquetage de l'outil de mise à jour
-10605	Erreur durant la copie du paquetage de l'outil de mise à jour
-10606	Impossible de désactiver l'outil de mise à jour actuel
-10607	Impossible de créer le répertorie d'installation du nouvel outil de mise à jour
-10608	Impossible de trouver le chemin de la mise à jour {path}
-10609	Impossible de démarrer l'outil de mise à jour
-10611	Impossible de démarrer l'outil de mise à jour (installation invalide)
-10612	L'application en cours d'exécution ne peut se mettre à jour elle-même
-10613	Impossible de créer 2 fenêtres formulaire de type barre d'outil

Erreurs du gestionnaire de fichiers du système (-124 -> -33)

Le tableau suivant liste les codes d'erreurs retournés par le gestionnaire de fichiers du système d'exploitation. Ces erreurs peuvent se produire en particulier lorsque vous utilisez les commandes du thème **Documents système**. Pour plus d'informations, reportez-vous à la section **Documents système**.

Code	Description
-124	Tentative d'accès à un volume partagé déconnecté
-121	Un chemin d'accès ne peut être créé
-120	Tentative d'accès à un fichier avec un chemin d'accès spécifiant un répertoire inexistant
-84	Problème physique avec le disque (installation ou formatage incorrect...)
-64	Problème physique avec le disque (installation ou formatage incorrect...)
-61	L'accès en lecture/écriture ne permet pas d'écrire.
-60	Mauvais secteur de répertoire principal. Votre disque est endommagé.
-58	Erreur de fichier système
-57	Tentative d'écriture sur un disque non-Macintosh
-54	Tentative d'écriture dans un fichier verrouillé
-53	Le volume a été éjecté.
-52	Erreur interne du gestionnaire de fichiers (le marqueur de fichiers est perdu).
-51	Tentative d'accès à un fichier avec un numéro de référence de fichier invalide
-49	Fichier déjà ouvert en Lecture/Ecriture.
-48	Tentative d'utilisation du nom d'un fichier déjà supprimé pour renommer un fichier.
-47	Fichier déjà ouvert, ou dossier non vide.
-46	Volume verrouillé par logiciel.
-45	Fichier verrouillé ou chemin d'accès invalide.
-44	Disque physiquement verrouillé.
-43	Fichier non trouvé.
-42	Trop de fichiers ouverts.
-41	Mémoire insuffisante pour ouvrir un nouveau fichier.
-40	Tentative de lecture ou d'écriture avant le début du fichier.
-39	Tentative de lecture après la fin de fichier.
-38	Tentative de lecture ou d'écriture dans un fichier non ouvert.
-37	Nom de fichier ou de volume incorrect. Il est trop long et/ou comporte un caractère
-36	Erreur d'Entrée/Sortie. Il y a probablement un secteur défectueux sur le disque.
-35	Le volume n'existe pas.
-34	Le disque est plein. Il n'y a plus de place disponible sur le disque.
-33	Le répertoire du disque est plein. Vous ne pouvez pas créer de fichier sur le disque.

Erreurs du gestionnaire de mémoire du système (-117 -> -108)

Le tableau ci-dessous liste les principaux codes d'erreurs retournés par le gestionnaire de mémoire du système d'exploitation.

Code	Description
-117	Problème interne de mémoire. La mémoire est probablement endommagée. Quittez l'application dès que possible. Redémarrez la machine et réouvrez la base.
-111	Problème interne de mémoire. La mémoire est probablement endommagée. Quittez l'application dès que possible. Redémarrez la machine et réouvrez la base. (*)
-109	Problème interne de mémoire. La mémoire est probablement endommagée. Quittez l'application dès que possible. Redémarrez la machine et réouvrez la base.
-108	Mémoire disponible insuffisante. Allouez davantage de mémoire à votre application 4D.

Conseil : Lorsque vous travaillez avec des gros tableaux, des BLOBs, des images ou des ensembles (c'est-à-dire des objets pouvant manipuler de grandes quantités de données), utilisez une méthode projet installée par **APPELER SUR ERREUR** pour tester l'erreur -108.

(*) Une erreur -111 peut également se produire lorsque vous tentez de lire une valeur dans un BLOB à un offset hors intervalle. Dans ce cas, cette erreur est mineure et vous n'êtes pas obligé de fermer la session de travail. Il vous suffit de corriger l'offset que vous avez passé à la commande BLOB.

Erreurs du gestionnaire d'impression du système (-8192 -> -1)

Le tableau ci-dessous liste les codes d'erreurs retournés par le gestionnaire d'impression du système d'exploitation. Ces codes peuvent être retournés durant l'impression.

Code	Description
-8192	Time out de l'imprimante
-8151	L'imprimante a été initialisée avec une version du driver différente de la vôtre
-8150	Aucune imprimante n'a été sélectionnée
-4101	Imprimante éteinte ou non connectée
-4100	La connexion avec l'imprimante a été interrompue
-193	Fichier de ressources introuvable
-128	Impression interrompue par l'utilisateur
-27	Problème de communication avec l'imprimante
-1	Problème lors de l'enregistrement d'un fichier à imprimer

Erreurs du gestionnaire de ressources du système (-196 -> -1)

Le tableau ci-dessous liste les codes d'erreurs retournés par le gestionnaire de ressources du système d'exploitation.

Code	Description
-196	La ressource ne peut être supprimée
-194	La ressource ne peut être ajoutée
-193	La ressource est endommagée (le fichier doit être réparé)
-192	Ressource introuvable
-1	Impossible d'ouvrir ce fichier de ressources

Erreurs SANE NaN (1 -> 255)

Le tableau ci-dessous liste les codes NaN retournés par le système d'exploitation. Le sigle NaN signifie “Not a Number”. C'est une représentation du Standard Apple Numeric Environment (SANE) qui est appelée lorsqu'une opération produit un résultat se trouvant dans le domaine de SANE.

Code	Description
1	Racine carrée invalide
2	Addition invalide
4	Division invalide
8	Multiplication invalide
9	Reste invalide
17	Conversion d'une chaîne ASCII invalide
20	Conversion d'un nombre de type Comp en virgule flottante
21	Création d'un NaN avec un code zéro
33	Argument invalide passé à une fonction trig
34	Argument invalide passé à une fonction trig inversée
36	Argument invalide passé à une fonction log
37	Argument invalide passé à une fonction xi ou xy
38	Argument invalide passé à une fonction financière
255	Stockage non initialisé

Erreurs du gestionnaire de son du système (-209 -> -203)

Le tableau ci-dessous liste les codes retournés par le gestionnaire de son du système d'exploitation.

Code	Description
-209	Le canal de son est indisponible
-207	Mémoire insuffisante pour jouer le son
-206	Format de ressource son incorrect
-205	Le canal de son est endommagé
-204	La ressource son ne peut être chargée
-203	Trop de commandes de sons

Erreurs du gestionnaire de port série du système (-28)

Le tableau ci-dessous fournit le code d'erreur retourné par le gestionnaire de port série du système d'exploitation.

Code	Description
-28	Il n'y a pas de port série ouvert

Erreurs Mac OS (4 -> 28)

Le tableau ci-dessous liste certaines erreurs courantes retournées par Mac OS. Lorsqu'une de ces erreurs se produit, il n'est généralement pas possible de poursuivre la session sans redémarrer.

Code	Description
4	Division par zéro
15	Erreur du chargeur de segments : 4D n'a pas pu charger un de ses propres segments de code. Vous devez allouer davantage de mémoire à 4D.
17 à 24	Un élément système est manquant. Vérifiez que votre dossier système a été correctement installé.
25	Mémoire saturée. Vous devez allouer davantage de mémoire à 4D.
28	La pile a été placée dans la heap de l'application. Vous devez allouer davantage de mémoire à 4D.

Erreurs diverses (-10700)

Le tableau suivant liste les erreurs diverses qui peuvent se produire :

Code	Description
-10700	Le port doit être compris entre 0 et 65535.

Codes de caractères

Codes Unicode

Dans les bases de données créées avec 4D v11, le langage ainsi que le moteur de la base de données stockent et manipulent nativement les caractères en Unicode.
Ce principe facilite l'internationalisation des applications 4D. L'Unicode est un jeu de caractères standard unifié qui gère pratiquement toutes les langues usuelles de la planète. Un jeu de caractères est une table de correspondance caractère/valeur numérique, par exemple "a"->1, "b"->2, "5"->15, "oe"->662, etc. Alors qu'en ASCII la valeur numérique de base est typiquement comprise entre 1 et 127, en Unicode la borne haute va au-delà de 65000, ce qui permet de représenter quasiment tous les caractères de toutes les langues.
Il existe différentes manières de coder les valeurs numériques Unicode : UTF-16 les code sur des entiers de 16-bits, UTF-32 sur des entiers de 32-bits et UTF-8 sur des entiers de 8-bits. 4D utilise principalement UTF-16 (comme Windows et Mac OS). Parfois, essentiellement pour des besoins liés au Web, 4D utilise UTF-8 qui a l'avantage de la compacité et de la lisibilité pour les caractères usuels (a-z,0-9).

Pour plus d'informations sur la norme Unicode, reportez-vous par exemple à la page suivante :
http://fr.wikipedia.org/wiki/Unicode
Une liste de codes Unicode (page en anglais) :
http://en.wikipedia.org/wiki/List_of_Unicode_characters

Attention : En unicode dans 4D v11, les codes de caractères suivants sont réservés et ne doivent jamais être inclus dans un texte:
0
65534 (FFFE)
65535 (FFFF)

Note de compatibilité : Les bases de données créées avec une version de 4D antérieure à la 11 peuvent fonctionner en mode compatibilité ASCII. Pour plus d'informations, reportez-vous à la section **EXPORTER TEXTE**.

Codes des touches de fonction

4D retourne le code clavier des touches de fonction dans la variable système *Keycode*, qui est utilisée dans les méthodes projet installées par la commande **APPELER SUR EVENEMENT** pour intercepter les événements.

Les valeurs des touches de fonctions ne sont pas basées sur des **EXPORTER TEXTE**. Elles sont listées ci-dessous :

Touche	Code
F1	-122
F2	-120
F3	-99
F4	-118
F5	-96
F6	-97
F7	-98
F8	-100
F9	-101
F10	-109
F11	-103
F12	-111
F13	-105
F14	-107
F15	-113

Par ailleurs, le tableau suivant liste les valeurs retournées dans la variable système *Keycode* lorsque vous appuyez sur les touches standard comme Retour chariot ou Entrée.

Touche	Code
Entrée	3
Retour chariot	13
Retour arrière	8
Tabulation	9
Echappement	27
Effacement	127
Aide	5
Début	1
Fin	4
Haut de page	11
Bas de page	12
Flèche gauche	28
Flèche droite	29
Flèche haut	30
Flèche bas	31

4D - Langage - Commandes obsolètes

16.0

- _o_ÉCRIRE RESSOURCE CHAINE
- _o_ÉCRIRE RESSOURCE IMAGE
- _o_ÉCRIRE RESSOURCE TEXTE
- _o_ENREGISTRER IMAGE
- _o_Fin sous enregistrement
- _o_FIXER EXECUTABLE CGI
- _o_FIXER INTERFACE
- _o_FIXER LIMITES AFFICHAGE WEB
- _o_FIXER TEMPORISATION WEB
- _o_GRAPHE SUR SELECTION
- _o_INACTIVER BOUTON
- _o_INTEGRER FICHIER HISTORIQUE
- _o_INVERSER FOND
- _o_ISO vers Mac
- _o_Lire ID ressource composant
- _o_Lire interface
- _o_LISTE SEGMENTS DE DONNÉES
- _o_LISTE TYPES IMAGES
- _o_Mac vers ISO
- _o_Mac vers Windows
- _o_MODIFIER SOUS ENREGISTREMENT
- _o_Nom de police
- _o_Numéro de police
- _o_Pendant
- _o_QT CHARGER ET COMPRESSER IMAGE
- _o_QT COMPRESSER FICHIER IMAGE
- _o_QT COMPRESSER IMAGE
- _o_REDESSINER LISTE
- _o_SOUS ENREGISTREMENT PRECEDENT
- _o_SOUS ENREGISTREMENT SUIVANT
- _o_Sous enregistrements trouvés
- _o_SUPPRIMER RESSOURCE
- _o_SUPPRIMER SOUS ENREGISTREMENT
- _o_TABLEAU ALPHA
- _o_TABLEAU VERS LISTE DE CHAÎNES
- _o_TOUS LES SOUS ENREGISTREMENTS
- _o_TRIER SOUS ENREGISTREMENTS
- _o_UTILISER BASE EXTERNE
- _o_UTILISER BASE INTERNE
- _o_Windows vers Mac
- _o_XSLT APPLIQUER TRANSFORMATION
- _o_XSLT FIXER PARAMETRE
- _o_XSLT LIRE ERREUR