4D v13

Mise à jour
Windows®/Mac OS®

4D v13 - Mise à jour
Versions Windows® and Mac OS®

Sommaire

1 Bienvenue

Bienvenue dans 4D v13. Cette nouvelle version est résolument placée sous le signe de l'ouverture, tout en préservant une compatibilité maximale avec les bases existantes :

- ouverture vers les autres bases de données et l'Internet grâce à un serveur HTTP entièrement réécrit, un tout nouveau client HTTP ou encore un pilote ODBC 64 bits,
- ouverture vers les autres outils de développement avec le nouveau *source toolkit*, un jeu de commandes complet (thème "Accès objets développement") fournissant un accès en lecture et en écriture à toutes les méthodes de la base, permettant par exemple d'exporter et de gérer l'intégralité de votre code sous forme de fichiers texte standard.
- ouverture vers les interfaces de nouvelle génération avec l'utilisation du moteur graphique Direct2D et du prévisualisateur d'impression XPS (sous Windows), le choix du moteur de rendu pour les zones Web, les nombreuses nouvelles options des listbox ou encore la possibilité d'indexer les métadonnées des champs image,
- enfin, 4D v13 propose également de nombreuses nouvelles fonctions destinées à optimiser et enrichir le développement des applications 4D : stockage externe des données volumineuses, nouvel ensemble de commandes complémentaires permettant d'accéder aux objets de formulaire par programmation.

Conversion des anciennes bases

Les bases de données créées avec des versions 6.x, 2003.x, 2004.x, 11.x et 12.x de 4e Dimension, 4D ou 4D Server sont compatibles avec 4D version 13 (fichier de structure et fichier de données).

- Les fichiers des bases de données en version 6.x, 2003.x, 2004.x doivent être convertis par l'intermédiaire d'un assistant et ne pourront plus être ouverts avec leur version d'origine.
- Les fichiers des bases de données en version 11 ou 12 sont convertis directement en version 13.

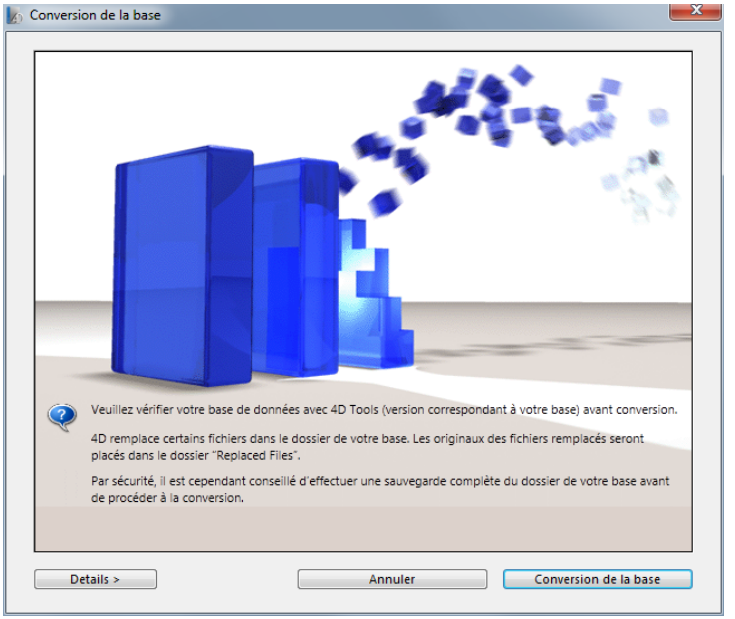
Note: Vous pouvez convertir tout fichier de structure interprété. Le fichier peut contenir le code compilé ; dans ce cas, il sera nécessaire de recompiler la base après conversion.

Bases en version 6.x, 2003.x et 2004.x

Les modifications structurelles apportées au niveau du moteur de la base de données de 4D nécessitent une conversion en profondeur de la structure et des données des bases antérieures à la version 11. Cette conversion s'effectue via un assistant spécifique. Par sécurité, l'assistant effectue une copie de la base d'origine avant sa conversion, de manière à ce que vous puissiez à tout moment revenir en arrière.

Il est conseillé de vérifier avant la conversion l'intégrité de la base à l'aide de l'utilitaire 4D Tools (compactage, vérification et, éventuellement, réparation de la base). Utilisez la version de 4D Tools correspondant à celle de la base d'origine.

Pour convertir une ancienne base, il suffit de la sélectionner dans la boîte de dialogue d'ouverture de 4D v13. L'assistant de conversion apparaît automatiquement :

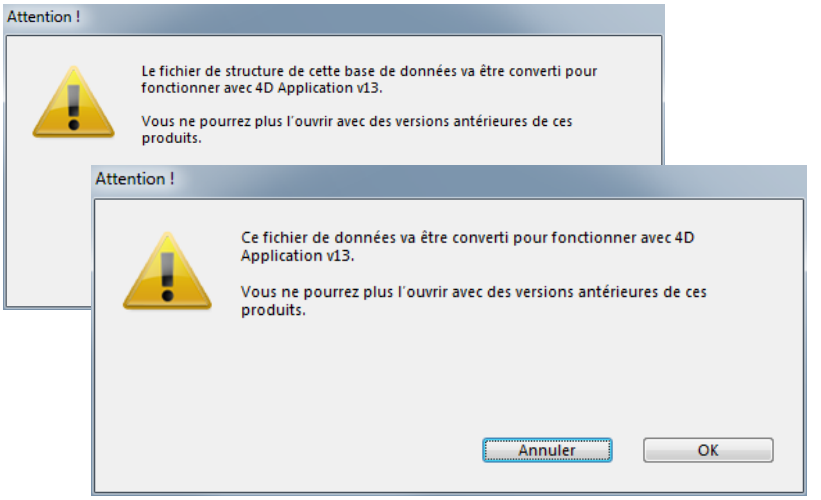


Cliquez sur le bouton **Conversion de la base** pour débuter le processus standard de conversion des fichiers de structure et de données.

- Pour plus d'informations sur cette boîte de dialogue et sur le processus de conversion, reportez-vous au paragraphe « Conversion des anciennes bases de données » dans le manuel *Mode Développement* de 4D.
- Certains mécanismes obsolètes ou anciens ne sont plus pris en charge et seront supprimés ou remplacés au cours de la conversion. Pour plus d'informations, reportez-vous au manuel *Mise à jour 4D v11 SQL*. Pour une description détaillée de toutes les modifications, reportez-vous au document *Guide de migration 4D v11 SQL* disponible en téléchargement à partir de l'adresse : http://doc.4d.com/home.fr.html

Bases en version 11 ou 12

La conversion d'une base en version 11 ou 12 s'effectue directement lors de l'ouverture du fichier de structure avec 4D v13. Des boîtes de dialogue d'alerte successives vous indiquent que les fichiers vont être convertis et qu'il ne sera plus possible de les ouvrir avec une version précédente :



A noter que la conversion du fichier de données entraîne la reconstruction des index.

Note: Si vous souhaitez convertir une base v11 utilisant une ancienne version de 4D Pack, il est recommandé d'installer avant conversion la version la plus récente du plug-in 4D Pack v11 dans la base d'origine.

Composants version 11 ou 12

4D v13 peut directement ouvrir des composants v12 ou v11 (compilés ou interprétés) sans conversion ni boîte de dialogue de confirmation. Rappelons que les composants sont toujours ouverts en lecture seule.

Il n'est pas nécessaire de recompiler les composants mais la conversion en v13 n'est possible que sur les fichiers .4DB et non .4DC.

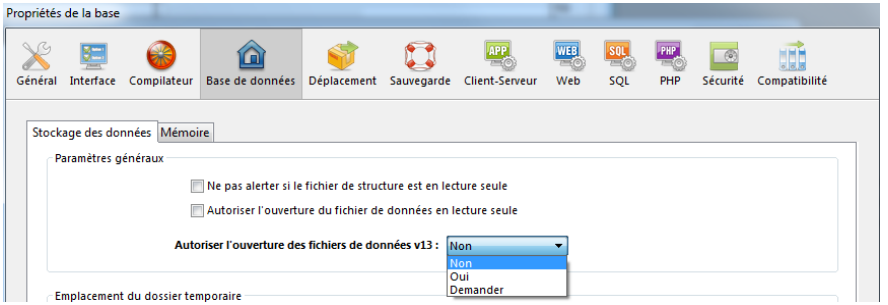
Rétro-compatibilité des bases converties

Le tableau suivant livre les scénarios de rétro-compatibilité des fichiers des bases de données 4D convertis en v13, c'est-à-dire la possibilité de les rouvrir avec des versions précédentes :

Versions 4D	Ouverture fichier de structure v13	Ouverture fichier de données v13
4D v11 et précédentes	non	non
4D v12	non	oui si autorisé(*)

(*) Il reste possible de rouvrir en version 12 un fichier de données converti avec 4D v13. Cette possibilité doit être autorisée explicitement côté version 12 à l'aide de l'option **Autoriser l'ouverture des fichiers de données v13** située dans la page "Base de données/Gestion des données" des Propriétés de la base :

Préférences de 4D v12
Option d'ouverture d'un fichier de données v13



Par défaut, l'option **Non** est sélectionnée. Sélectionnez **Oui** pour autoriser l'ouverture directe des données et **Demander** pour afficher au préalable une boîte de dialogue de confirmation.

Cette option, destinée à permettre la récupération de données dans des cas spécifiques, doit être utilisée avec précautions :

- avant la réouverture des données : dans tous les cas, il est recommandé de supprimer le fichier d'index v13 (fichier .4dindx).
- après la réouverture des données : dans le cas où des nouvelles fonctions de la version 13 ont été appliquées aux tables de la base, il est recommandé de compacter ou réparer le fichier de données à l'aide du CSM.

Compatibilité fonctionnelle

Cette section décrit les nouveautés de 4D v13 pouvant modifier le fonctionnement de vos applications converties depuis des versions précédentes.

4D Chart devient un plug-in externe

Le plug-in 4D Chart est retiré de l'application 4D v13 et est désormais uniquement proposé en tant que plug-in additionnel standard. Dans les versions précédentes de 4D, ce plug-in était intégré à l'application.

Si vos applications 4D utilisaient des fonctions de 4D Chart, il vous suffit d'installer le plug-in externe 4D Chart v13 dans votre environnement de travail afin de conserver leur fonctionnement. L'installation s'effectue comme pour tout plug-in, via un simple glisser-déposer du dossier du plug-in dans le dossier **Plugins** de 4D ou de votre base.

À noter que ce plug-in n'évoluera plus dans les prochaines versions de 4D, il est recommandé dès à présent de privilégier l'emploi du SVG pour la représentation graphiques des données.

Fonctions obsolètes

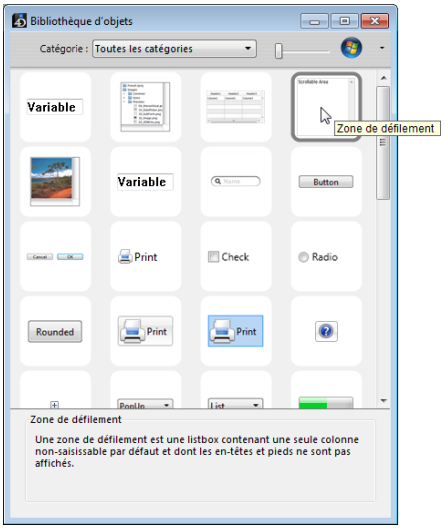
AP Get file MD5 digest (4D Pack)

La fonction de 4D Pack **AP Get file MD5 digest** est maintenue par compatibilité mais est désormais obsolète et ne doit plus être utilisée. Elle doit être remplacée par la nouvelle routine Generer digest, page 172.

Zone de défilement

Il n'est plus possible de créer des objets de type "Zone de défilement" dans l'éditeur de formulaires de 4D. Dans les nouvelles interfaces, ces objets sont avantageusement remplacés par des list box.

Dans 4D v13, vous pouvez créer facilement des zones de défilement basées sur des list box à l'aide du nouvel objet "Zone de défilement" de la bibliothèque d'objets préconfigurée :



Note: Dans les bases de données converties, les zones de défilement existantes sont converties en list box et une option **Compatibilité** est automatiquement cochée dans les "Propriétés de la base" afin de leur assurer un fonctionnement identique à la version précédente (cf. paragraphe "Compatibilité Zones de défilement", page 45). Il est toutefois recommandé de remplacer ces list box en mode compatibilité par des list box standard.

Affichage des symboles Mac OS

Dans 4D v13, il n'est plus possible de visualiser les anciens symboles Mac OS dans l'éditeur de méthodes :

- † (Option+t) pour les constantes heure
- ≤...≥ (Option+< et Option+>) pour les symboles d'indice de chaîne.

Sous Mac OS, vous pouvez continuer d'insérer ces symboles en utilisant les combinaisons de touches associées mais leurs équivalents multi-plates-formes (?, [[et]]) seront affichés et stockés.

Fonctions supprimées

Comme annoncé lors de la sortie de la version 12 de 4D, certaines fonctions obsolètes sont supprimées de la gamme 4D à compter de 4D v13.

Mode contextuel du serveur Web

Le mode contextuel n'est pas pris en charge dans le nouveau serveur Web de 4D v13. Toutes les options, constantes ou commandes relatives au mode contextuel (par exemple Contexte Web, FIXER LIMITES AFFICHAGE WEB ou FIXER TEMPORISATION WEB) sont désactivées. Ces commandes sont désormais préfixées "_o_" et leur exécution génère l'erreur 33 (*Méthode ou fonction non implémentée*).
Pour des fonctionnalités approchantes, nous vous invitons à étudier le nouveau mécanisme des sessions Web (cf. paragraphe "Gestion des sessions Web", page 67).

Support des CGI externes

Le serveur HTTP de 4D v13 ne prend plus en charge l'exécution de CGI externes, en mode automatique (appel via l'URL) ou manuel (appel via la commande FIXER EXECUTABLE CGI). La commande FIXER EXECUTABLE CGI a été préfixée "_o_" et son exécution génère l'erreur 33 (*Méthode ou fonction non implémentée*).

Points d'arrêt provisoires

Les points d'arrêt provisoires ne sont plus disponibles dans 4D v13. Ils sont avantageusement remplacés par le nouveau Menu contexuel du débogueur.

Motifs monochromes

L'ancienne palette de motifs de l'éditeur de formulaires, utilisée à l'origine pour les interfaces monochromes, a été supprimée. Dans les bases de données converties, les motifs existants sont automatiquement remplacés par des combinaisons de couleurs. Pour plus d'informations, reportez-vous au paragraphe "Conversion des anciens motifs", page 48).

Configuration minimale

Les applications de la gamme 4D version 13 requièrent au minimum les configurations suivantes :

	Windows	Mac OS
Processeur	Intel® Core Duo	
Système	Windows XP SP3(*), Windows 7	Mac OS version 10.6.8 et ultérieure
Mémoire RAM	2 Go	
Résolution écran	1280 * 1024 pixels	

(*) Le moteur de rendu intégré WebKit n'est pas pris en charge par 4D v13 sous Windows XP. Pour plus d'informations, reportez-vous au paragraphe "Choix du moteur de rendu Web", page 50.

2 Base de données

De nouvelles fonctions et optimisations sont proposées dans le moteur de base de données de 4D v13 :

- prise en charge d'index de mots-clés sur les champs image,
- mémorisation des noms de fichiers image importés,
- possibilité de stocker les données des champs BLOB, Texte et Image à l'extérieur du fichier de données.

Champs image

Ce paragraphe présente les nouveautés liées à la gestion des champs image dans 4D v13.

Note Il est également possible désormais de stocker les données des champs image en-dehors du fichier de données. Ce point est traité dans le paragraphe "Stockage externe des données", page 24.

Indexation des mots-clés d'images

Depuis 4D v12, il est possible de stocker et de gérer les métadonnées associées aux champs image, notamment à l'aide des commandes FIXER METADONNEES IMAGE et LIRE METADONNEES IMAGE.

4D v13 vous permet d'optimiser l'accès à ces métadonnées en les indexant. Grâce à cette fonction, la recherche d'images par mots-clés stockées peut s'effectuer de manière quasi instantanée.

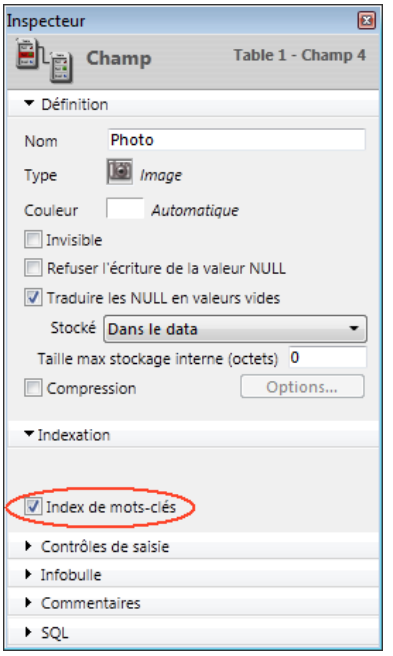
Métadonnées indexables

Les index de mots-clés d'images sont exclusivement basés sur les métadonnées de type **IPTC/Keywords**. Ces types de métadonnées sont notamment pris en charge par les formats d'image TIFF et JPEG (à noter que BMP, PNG et GIF ne les prennent pas en charge). Les autres types de métadonnées ne sont pas pris en compte par l'indexation.

Gestion des index de mots-clés d'images

La création ou la suppression d'un index de mots-clés associé à un champ image s'effectue comme pour un index de mots-clés standard. Vous pouvez effectuer l'opération depuis l'éditeur de structure ou par programmation.

- **Depuis l'Inspecteur de l'éditeur de Structure** : vous pouvez créer ou supprimer l'index en cochant ou en désélectionnant l'option correspondante dans l'Inspecteur pour un champ image.



Note Vous pouvez également créer l'index à l'aide du menu contextuel de l'éditeur en choisissant la commande **Index>Mots-clés**.

- **Depuis l'Explorateur d'index de l'éditeur de Structure** : dans l'Explorateur d'index (accessible via un clic sur le bouton  ou la commande **Nouvel index...** du menu contextuel), vous pouvez visualiser les champs image et les ajouter aux index existants. Seul le type d'index Mots-clés est disponible pour les champs image.
- **Par programmation** : les commandes existantes CREER INDEX et SUPPRIMER INDEX prennent en charge les index de mots-clés des champs image.

Note Il est possible d'utiliser la commande FIXER INDEX (avec le paramètre *index* = -1) pour créer un index de mots-clés image. Toutefois, dans ce cas vous ne pourrez pas le supprimer par programmation.

La mise à jour des index de mots-clés d'images est effectuée automatiquement par 4D à chaque sauvegarde du champ image (création ou modification d'enregistrement, importation de données, etc.). Les métadonnées de type **IPTC/Keywords** sont automatiquement indexées par 4D lorsqu'elles sont présentes dans l'image (il n'est pas nécessaire d'appeler la commande FIXER METADONNEES IMAGE pour les inclure dans l'index du champ image).

Utilisation des index de mots-clés d'image

L'utilisation des index de mots-clés d'image peut accélérer de façon importante vos applications.

Comme pour les index de mots-clés des champs texte, vous exploitez ces index à l'aide de l'**opérateur %** : cet opérateur doit être placé dans les formules de recherche ou de tri afin d'utiliser spécifiquement une valeur de l'index. Par exemple :

```
CHERCHER ([IMAGES];[IMAGES]Photos % "cats")
  // cherche les photos associées au mot-clé cats
```

Ce principe est valide pour toutes les commandes de recherche et de tri : CHERCHER PAR FORMULE, CHERCHER DANS SELECTION, TRIER, etc. Pour plus d'informations sur le fonctionnement de l'opérateur %, reportez-vous à la section "Opérateurs de comparaison" dans le manuel *Langage* de 4D.

Liste des mots-clés

Vous pouvez obtenir la liste de tous les mots-clés d'image définis pour un champ et la liste des mots-clés associés à une image.

- Comme pour les champs texte, vous pouvez obtenir la liste des mots-clés contenus dans l'index des mots-clés d'un champ image en utilisant la commande VALEURS DISTINCTES.
Lorsqu'elle est appliquée à un champ image associé à un index de mots-clés, la commande remplit le tableau passé en paramètre avec les mots-clés de l'index.

Note Si le champ image n'est pas associé à un index de mots-clés, le tableau est retourné vide.

Par exemple :

```
TOUT SELECTIONNER([IMAGES])
TABLEAU TEXTE(<>_MesMotsCles;10)
VALEURS DISTINCTES([IMAGES]Photos;<>_MesMotsCles)
```

- Pour obtenir la liste des mots-clés associés à une image, vous pouvez utiliser la nouvelle commande LIRE MOTS CLES IMAGE, page 146.

Noms par défaut des fichiers image

4D v13 peut désormais mémoriser un nom par défaut pour chaque image stockée dans un champ. Ce principe permet de proposer un nom de fichier par défaut lors de l'écriture du contenu du champ image dans un fichier disque via un export utilisateur ou la commande ECRIRE FICHIER IMAGE (lorsque vous passez une chaîne vide dans le paramètre nomFichier).

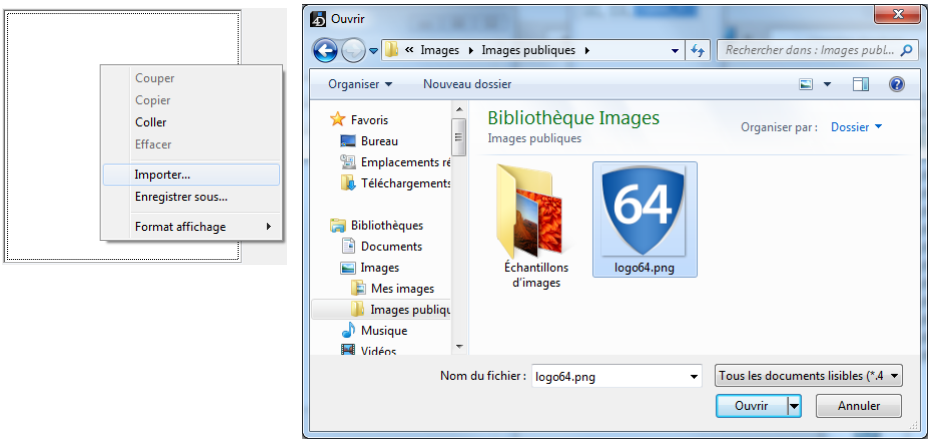
Si le contenu du champ est copié dans une variable ou dans un autre champ, le nom par défaut est également copié.

L'association d'un nom par défaut à une image stockée dans un champ image s'effectue de deux manières :

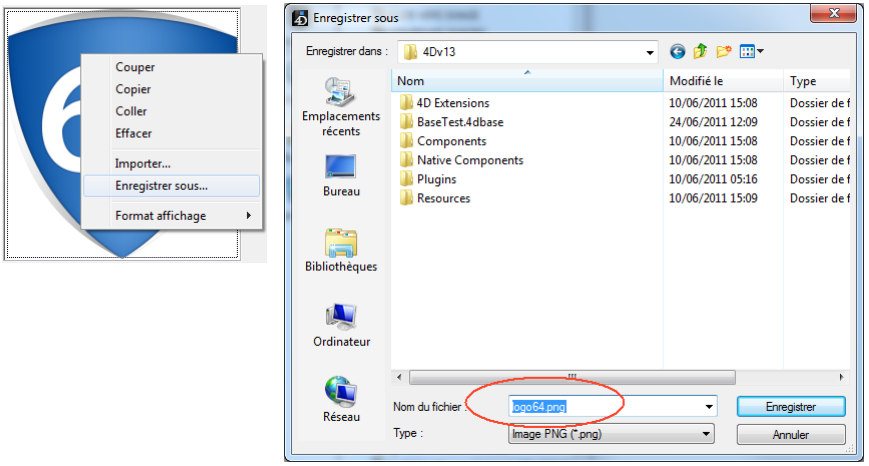
- par programmation, à l'aide de la nouvelle commande FIXER NOM FICHIER IMAGE, page 144. Cette commande permet d'associer un nom de fichier par défaut à l'image. La nouvelle commande Lire nom fichier image, page 147 permet de connaître le nom par défaut d'une image.
- automatiquement, lorsque le contenu d'un fichier image est importé dans un champ image via le menu contextuel ou via la commande LIRE FICHIER IMAGE : dans ce cas, 4D mémorise le nom du fichier image d'origine.

Ce principe est illustré dans la séquence suivante :

- 1. L'utilisateur importe le fichier nommé *logo64.png* dans un champ image :



- 2. Par la suite, l'utilisateur enregistre le contenu du champ image : le nom *logo64.png* est proposé dans la boîte de dialogue (à noter qu'il s'agit d'un nom par défaut, il peut être modifié) :



Stockage externe des données

Dans 4D v13, vous pouvez choisir le lieu de stockage des données de chaque champ de type BLOB, Image et Texte. En plus des options de stockage existantes (dans l'enregistrement ou dans le fichier de données), vous pouvez choisir de stocker ces champs en-dehors du fichier de données. Dans ce cas, vos données sont enregistrées sous forme de fichiers externes qui sont gérés par 4D peuvent être manipulés par des applications tierces - à condition de procéder avec précaution afin de préserver l'intégrité des données.

Ce principe est transparent pour l'utilisateur : l'accès aux données est identique quel que soit leur emplacement.

Le stockage externe des données permet d'optimiser le fonctionnement de l'application en permettant par exemple de déporter les données les plus volumineuses ou d'accéder depuis le système d'exploitation aux textes ou aux images contenu(e)s dans la base, même lorsqu'elle n'est pas ouverte.

Mode automatique ou personnalisé

La prise en charge du stockage externe des données peut être effectuée dans deux modes :

- **Mode automatique** : dans ce mode, 4D crée et gère un dossier par défaut, structuré de façon spécifique et contenant toutes les données à stockage externe. Dans ce cas, la gestion de ces données s'effectue de manière transparente, vous bénéficiez des mêmes fonctions et automatismes que pour les champs stockés en interne.
  Pour activer le mode automatique, vous pouvez soit :
  - Configurer le stockage externe dans l'éditeur de structure (paramétrage enregistré avec la structure de la base),
  - Utiliser la nouvelle commande FIXER CHEMIN DONNEES EXTERNES avec une chaîne vide dans le paramètre *chemin* (paramétrage valide pour la session).
- **Mode personnalisé :** dans ce mode, vous désignez librement le lieu de stockage des fichiers externes pour chaque champ et chaque enregistrement. Dans ce cas, 4D conserve uniquement le lien entre le champ et ses données, certains mécanismes de base de données ne sont plus disponibles.
  Le mode personnalisé est activé à l'aide de la nouvelle commande FIXER CHEMIN DONNEES EXTERNES en désignant un chemin d'accès complet (autre que le dossier par défaut).

Le tableau suivant compare les fonctions et automatismes disponibles dans les modes automatique et personnalisé :

	Mode automatique	**Mode personnalisé**
Définition du dossier de stockage	Nom et emplacement définis par 4D ; un seul dossier par défaut pour toute la base	Nom et emplacement libres, peut être différent pour chaque champ
Création, chargement et enregistrement des fichiers externes	Automatique	Automatique
Suppression du fichier externe si suppression de l'enregistrement	Oui	Non
Suppression du fichier externe si valeur Null affectée au champ	Oui	Non
Intégration automatique à la sauvegarde de la base	Oui	Non
Prise en charge automatique lors de l'intégration de l'historique	Oui	Non
Prise en charge des index standard (champs texte)	Non	Non
Prise en charge index de mots-clé (champs texte et image)	Oui	Non
Prise en charge des transactions	Oui	Non

Emplacement des données externes

Les données stockées à l'extérieur du fichier de données sont organisées selon les principes suivants :

- Pour chaque enregistrement, les données sont stockées dans un fichier externe nommé *xxx*.txt (champ Texte), *xxx*.blob (champ Blob) ou *xxx*.jpg, *xxx*.tiff... (champ Image, l'extension dépend du type de l'image) où *xxx* est un identifiant unique (UUID) géré par 4D.
- En mode automatique, toutes les données externes sont placées dans un dossier nommé **<*NomDeLaBase*>.ExternalData** où <*NomDeLaBase*> est remplacé par le nom du fichier de structure de la base. Ce dossier est placé à côté du fichier de données de la base (fichier .4DD).
A l'intérieur de ce dossier, 4D crée un dossier pour chaque table disposant de données externes (nommé "Table+*numéro de la table*"), puis un sous-dossier pour chaque champ externe (nommé "Field+*numéro du champ*)". Les 100 premiers éléments du champ sont stockés au premier niveau de ce dossier, les suivants sont stockés dans des sous-dossiers numérotés à partir de "2", chaque sous-dossier contenant 100 éléments. Par exemple, si le champ n°5 de la table n°3

de la base "Accounting" est stocké en-dehors des données, l'arborescence suivante sera créée par 4D :

```
Accounting.4DD
[Accounting.ExternalData]
   Table3
      Field 5
         Data_1B7F3A 56F6544B45951EFA60426D5ABC.txt
         Data_1B7F3A 56F6544B45951EFA60426D5CCC.txt
         ...
         2
            Data_2ADBFBA478AAE4409DA9C2D13C90A53B.txt
            Data_32F8A30B87EE7E4BBC802468D553DC43.txt
            ...
```

- En mode personnalisé, le nom et l'emplacement du dossier sont libres, ils peuvent être définis séparément pour chaque champ à stockage externe via la nouvelle commande FIXER CHEMIN DONNEES EXTERNES. Ce paramétrage n'est pas stocké dans la structure de la base.

Création et mise à jour des fichiers

L'écriture du champ dans un fichier externe est effectuée au moment où l'enregistrement est stocké sur disque (après validation de la transaction le cas échéant) :

- si le fichier externe n'existe pas, il est créé ;
- si un fichier externe existe déjà, il est écrasé par 4D.
  Si vous souhaitez le conserver, vous pouvez soit définir un autre chemin (à l'aide de la commande FIXER CHEMIN DONNEES EXTERNES), soit utiliser la nouvelle commande RECHARGER DONNEES EXTERNES afin de charger en mémoire le contenu du champ depuis son fichier externe avant de le réécrire sur disque. Cette possibilité est utile lorsque le fichier a été modifié par une autre application après le chargement de l'enregistrement.

Synchronisation et réplication

Le lieu de stockage des données est un paramètre local à chaque base. Dans le cadre d'une synchronisation ou d'une réplication, ces paramètres peuvent différer entre la base locale et la base distante. Dans ce cas, le stockage est réalisé conformément aux paramètres de chaque base, la synchronisation ou la réplication ne les modifient pas.

Par exemple, si un champ image de la base distante est stocké en-dehors du fichier de données et que le même champ de la base locale est stocké dans le fichier de données, il sera bien stocké dans le fichier de données après une opération de synchronisation.

Accès externes aux fichiers

Les fichiers de stockage externes sont accessibles en lecture écriture pour d'autres applications que 4D (système d'exploitation, éditeur de texte ou graphique, etc.). Toutefois, ces accès doivent être effectués avec précaution car ils peuvent altérer le fonctionnement de l'application :

- si un fichier de stockage externe est supprimé, renommé ou déplacé par le système d'exploitation ou application tierce, 4D considérera que le champ correspondant a la valeur Null et le fichier sera recréé (s'il n'est pas Null) au moment de la sauvegarde de l'enregistrement.
- si vous utilisez des index et si des fichiers de stockage sont modifiés par une application tierce sans que les enregistrements parents ne soient réécrits sur disque, les index ne seront pas mis à jour.

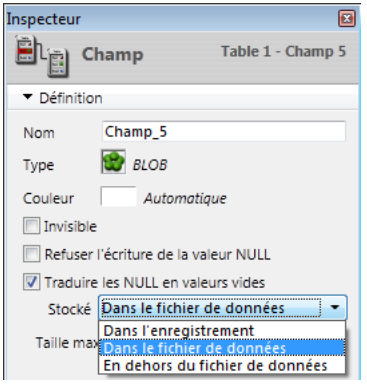
Note Les fichiers Texte externes sont stockés au format UTF-8 sans BOM. S'ils sont ouverts par une application tierce puis enregistrés avec une BOM, ils pourront être rouverts par 4D mais seront réenregistrés sans BOM.

Lecture seule des enregistrements et des fichiers

Le chargement en "lecture seulement" d'un enregistrement ne verrouille pas les fichiers externes des champs de l'enregistrement. Les fichiers restent modifiables sur disque par 4D ou des applications tierces, alors que leur contenu est chargé en mémoire par 4D.

Configurer le stockage externe dans l'éditeur de structure

Vous pouvez définir un stockage externe pour les champs de type BLOB, Image et Texte via l'éditeur de structure de 4D. Dans ce cas, le lieu de stockage est défini de façon permanente et les mécanismes d'accès aux données seront entièrement gérés par 4D (mode automatique).
Vous définissez cette option via un nouveau pop up menu dans l'Inspecteur de structure :



Vous disposez des options suivantes :

- **Dans l'enregistrement** et **Dans le fichier de données** : ces options correspondent aux options des versions précédentes de 4D.
  - **Dans l'enregistrement** : option auparavant disponible pour les champs de type Texte uniquement.
    Cette option est sélectionnée par défaut pour les champs de type Texte créés en version 13 car ce type de stockage est nécessaire si vous souhaitez utiliser des index "classiques" de type B-Tree.
  - **Dans le fichier de données** : lieu de stockage standard pour les champs Blob et Image (et optionnellement pour les champs Texte) dans les versions précédentes de 4D.
    Cette option est sélectionnée par défaut pour les champs Blob et Image.
- **En-dehors du fichier de données** : cette nouvelle option permet de stocker les données dans des fichiers séparés, en-dehors du fichier .4DD. Ce point est détaillé dans le paragraphe "Emplacement des données externes", page 25.

Note de compatibilité

Cette option est prise en compte uniquement pour les nouveaux enregistrements créés ultérieurement dans la table. Lorsque vous définissez cette option pour une table dans laquelle des enregistrements ont déjà été saisis, ils ne sont pas modifiés et la table travaillera en mode de stockage mixte interne/externe.
Si vous souhaitez étendre ce mode aux enregistrements existants, vous devez compacter les données avec la nouvelle option **Forcer la mise à jour des enregistrements** (cf. paragraphe "Centre de Sécurité et de Maintenance", page 60).

Taille max stockage interne

Pour tous les types de champs, l'option **Taille max stockage interne** est disponible lorsque vous sélectionnez **dans le fichier de données** ou **en-dehors du fichier de données**. Si votre application traite des données BLOB, Image ou Texte de tailles diverses, il peut être intéressant de moduler le lieu de stockage en fonction de ce critère afin d'optimiser ses performances.

L'option fonctionne comme dans les versions précédentes de 4D : la valeur saisie dans la zone représente la taille en octets au-dessous de laquelle les données du champ seront stockées dans l'enregistrement -- quel que soit le lieu de stockage défini.

Par exemple, si vous saisissez 30 000 pour un champ image, une image de 20 ko sera stockée dans l'enregistrement et une image de 40 ko sera stockée à l'emplacement défini (**dans le fichier de données** ou **en-dehors du fichier de données**). Par défaut, la valeur est 0 : toutes les données sont stockées en-dehors des enregistrements.

3 Atelier de développement

Cette section présente les nouveautés et modifications apportées à l'atelier de développement de 4D v13, tant au niveau des formulaires et de la génération d'applications que des outils mis à disposition des développeurs 4D :

- Extension des fonctions des list box,
- Modernisation des palettes de motifs,
- Intégration du Web Kit dans les zones Web,
- Nouveau moteur de rendu graphique Direct2D sous Windows,
- Utilisation de XPS pour les aperçus d'impressions sous Windows,
- Modification de la recherche des dépendances,
- Nouveau menu contextuel du débogueur,
- Nouvelles options dans le CSM,
- Nouveau format d'historique pour le générateur d'applications,
- Possibilité d'externaliser les propriétés utilisateur.

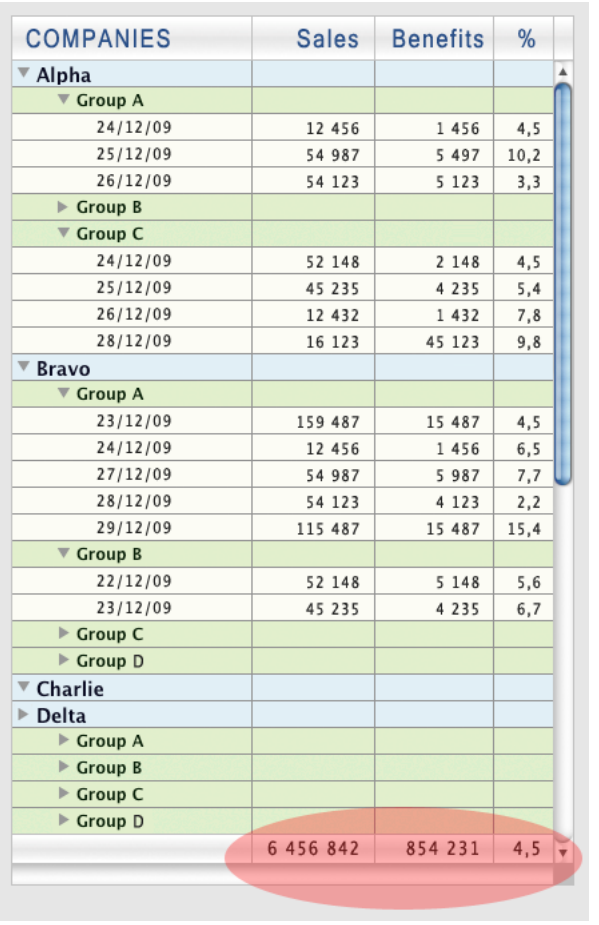
List box

Dans 4D v13, les list box ont bénéficié de multiples nouvelles fonctions permettant d'étendre largement leur champ d'application.

Note De nombreuses nouveautés sont également accessibles via les commandes de langage des list box. Pour plus d'informations, reportez-vous au paragraphe "List Box", page 149.

Pieds de page

Dans 4D v13, les List box peuvent contenir des zones de "pied de page" non saisissables affichant des informations supplémentaires. Dans les données présentées sous forme de tableaux, les pieds de page sont généralement utilisés pour afficher des calculs tels que des sommes ou des moyennes. Voici un exemple de list box utilisant un pied de page :



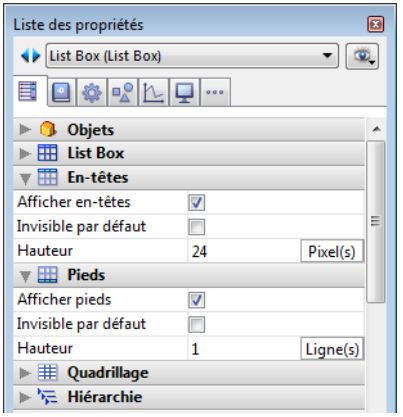
COMPANIES	Sales	Benefits	%
Alpha			
Group A			
24/12/09	12 456	1 456	4,5
25/12/09	54 987	5 497	10,2
26/12/09	54 123	5 123	3,3
Group B			
Group C			
24/12/09	52 148	2 148	4,5
25/12/09	45 235	4 235	5,4
26/12/09	12 432	1 432	7,8
28/12/09	16 123	45 123	9,8
Bravo			
Group A			
23/12/09	159 487	15 487	4,5
24/12/09	12 456	1 456	6,5
27/12/09	54 987	5 987	7,7
28/12/09	54 123	4 123	2,2
29/12/09	115 487	15 487	15,4
Group B			
22/12/09	52 148	5 148	5,6
23/12/09	45 235	4 235	6,7
Group C			
Group D			
Charlie			
Delta			
Group A			
Group B			
Group C			
Group D			
	6 456 842	854 231	4,5

Accès aux zones de pied de page

Les zones de pied sont disponibles pour tous les types de list box, sélections ou tableaux, hiérarchiques ou non-hiérarchiques.

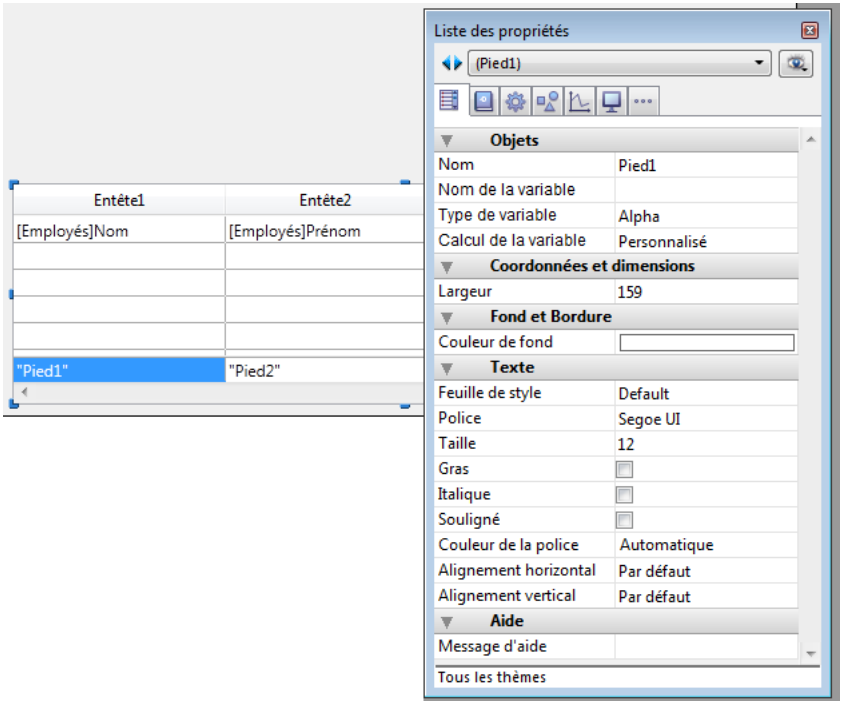
Attention : comme les zones d'en-têtes, les zones de pied ne sont pas saisissables en utilisation. Leur contenu est uniquement calculé.

La zone de pied est une nouvelle zone spécifique de list box. Pour pouvoir l'utiliser, vous devez cocher l'option **Afficher pieds** dans la List des propriétés, lorsqu'une list box est sélectionnée :



Vous disposez d'une zone de pied par colonne, configurable séparément.

Une fois la zone affichée au niveau de la list box, vous pouvez la sélectionner en cliquant dessus afin d'accéder à ses propriétés dans la Liste des propriétés :



Propriétés des zones de pied

Les zones de pied peuvent être définies via la Liste des propriétés ou des commandes du langage (cf. commande LISTBOX FIXER CALCUL PIED, page 149).

Les propriétés des zones de pied sont les suivantes :

- **Nom** et **Nom de la variable** : la zone de pied est un objet spécifique disposant d'un nom d'objet (devant être unique dans la page) et d'une variable associée. Par défaut, le nom de la variable est vide, 4D utilise des variables dynamiques.
- **Type de variable** : ce menu permet de préciser le type de la variable et de mettre à jour les options de la Liste des propriétés. À noter que si vous utilisez une variable non dynamique (vous nommez la variable), le typage de la variable devra être effectué via le langage.
- **Calcul de la variable** : cette option permet de définir le type de calcul à effectuer dans la zone de pied. Vous disposez de plusieurs types de calculs ainsi que l'option **Personnalisé**.

- **Minimum**, **Maximum**, **Somme**, **Nombre**, **Moyenne**, **Ecart type**, **Variance** et **Somme des carrés.** Ces calculs sont décrits dans le paragraphe “Calculs automatiques”, page 35. Lorsqu’un calcul est sélectionné, il est automatiquement appliqué à l’ensemble des valeurs présentes dans la colonne de la list box. A noter que le calcul ne tient pas compte du statut affiché/masqué des lignes de la list box. Si vous souhaitez restreindre un calcul aux lignes visibles, vous devez utiliser un calcul personnalisé.
  Lorsqu’un calcul automatique a été affecté à une zone de pied, un badge "action standard" lui est associé :

- **Personnalisé** : lorsque vous sélectionnez cette option, aucun calcul automatique n’est effectué par 4D. Vous devez affecter par programmation la valeur de la variable de la zone.

- **Coordonnées et dimensions**, **Affichage**, **Fond et Bordure**, **Texte** : ces propriétés sont identiques à celles des colonnes (hormis la couleur alternée).

- **Aide** : chaque zone de pied peut avoir son propre message d’aide.

Calculs automatiques

Vous pouvez associer divers calculs automatiques à une zone de pied. Le tableau suivant liste les calculs utilisables en fonction du type de données présentes dans la colonne :

	Numérique	Texte	Date	Heure	Booléen	Image
Minimum	X		X	X	X	
Maximum	X		X	X	X	
Somme	X			X	X	
Nombre	X	X	X	X	X	X
Moyenne	X			X		
Ecart type(*)	X			X		
Variance(*)	X			X		
Somme des carrés(*)	X			X		

(*) Uniquement pour les list box de type tableau.

Nouvel événement formulaire Sur clic pied

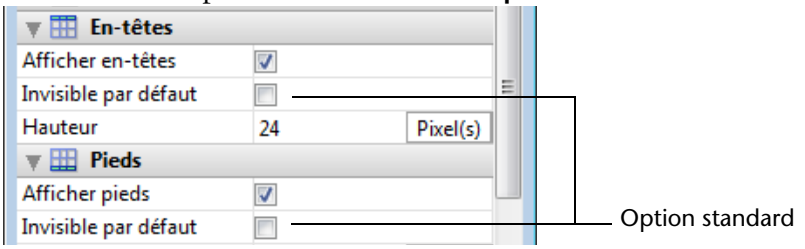
Un nouvel événement formulaire est disponible pour les objets List box et colonne de list box : **Sur clic pied**.

Cet événement est généré lorsqu'un clic se produit sur la zone de pied d'une list box ou d'une colonne de list box. Il peut être utilisé pour afficher par exemple un menu contextuel ou effectuer toute action personnalisée.

Note Voir aussi la description de la commande Evenement formulaire, page 136.

Invisible par défaut

Les zones d'en-tête (ainsi que les nouvelles zones de pied) bénéficient désormais de l'option standard **Invisible par défaut** :



Comme pour tous les objets de formulaire, cette option facilite la gestion de l'affichage de l'objet à l'aide de la commande OBJET FIXER VISIBLE.

A noter que l'option "Invisible par défaut" n'est disponible que lorsque l'option "Afficher en-têtes" ou "Afficher pieds" correspondante est cochée. La commande OBJET FIXER VISIBLE n'aura aucun effet si l'option d'affichage de la zone d'en-tête ou de pied correspondante n'a pas été cochée dans la liste des propriétés.

Hauteur des lignes, en-têtes et pieds

4D v13 propose deux nouveautés majeures relatives à la définition des hauteurs de lignes dans les list box :

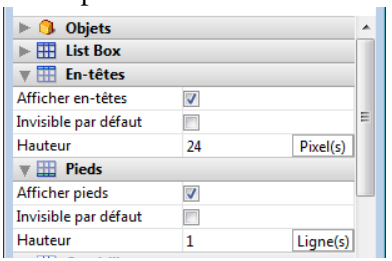
- il est désormais possible de définir individuellement la hauteur des en-têtes et des pieds ;
- il est désormais possible de définir une hauteur en "lignes" et non plus uniquement en pixels.

Hauteur des en-têtes et des pieds

Vous pouvez définir séparément la hauteur de la ligne d'en-tête et de la ligne de pied d'une list box. Ce paramétrage peut être effectué de deux manières :

- à l'aide de nouvelles propriétés dans la Liste des propriétés (cf. ci-dessous) ;
- à l'aide des nouvelles commandes LISTBOX FIXER HAUTEUR ENTETES et LISTBOX FIXER HAUTEUR PIEDS (cf. chapitre "Langage").

La liste des propriétés d'un objet list box comporte deux nouveaux thèmes permettant de configurer l'affichage et la hauteur des en-têtes et des pieds :



Note Vous pouvez choisir l'unité (lignes ou pixels) de la valeur de hauteur. Pour plus d'informations sur l'unité "lignes", reportez-vous au paragraphe suivant.

Par défaut, la hauteur des en-têtes et des pieds est de **1 ligne**. Cette valeur est également appliquée dans les bases de données converties depuis une version précédente de 4D.

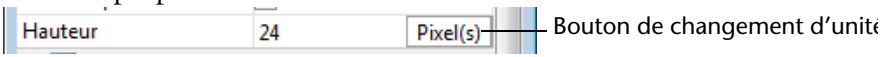
La hauteur minimum en pixels des en-têtes dépend du système. Si vous passez une valeur trop petite, elle sera remplacée par la taille minimum définie dans le système pour les en-têtes. Il n'y a pas de taille minimum pour les pieds et les lignes.

Note de compatibilité Sous Windows 7 et Windows Vista, la hauteur minimale des en-têtes est de 24 pixels. Les en-têtes de hauteur inférieure éventuellement définis dans vos bases converties seront automatiquement redimensionnés. Dans ce cas, il vous sera peut-être nécessaire de retoucher vos formulaires.

Hauteur définie en lignes

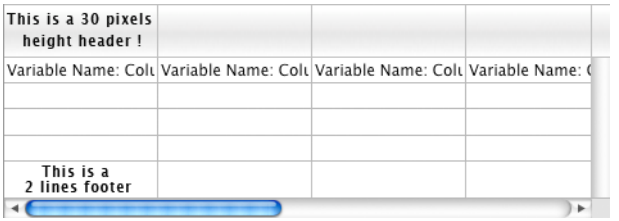
Dans les versions précédentes de 4D, la hauteur des lignes pouvait être définie en pixels uniquement. Désormais, il est possible de définir une hauteur en **lignes** de texte pour les en-têtes, pieds et lignes.

Le choix de l'unité est effectué à l'aide d'un paramètre spécifique pour les nouvelles commandes LISTBOX FIXER FORMULE COLONNE, LISTBOX FIXER HAUTEUR PIEDS et la commande modifiée LISTBOX FIXER HAUTEUR LIGNES ou via le bouton associé au champ "Hauteur" dans la Liste des propriétés :



Note Pour ne pas avoir à cliquer sur le bouton, vous pouvez insérer directement la lettre "L" ou "P" dans la zone de valeur pour l'unité Ligne ou Pixels (par exemple "17 P"). Le libellé du bouton est mis à jour.

Vous pouvez mixer les deux unités dans une même listbox :



- Lorsque vous utilisez l'unité "Pixels", la valeur de hauteur est directement appliquée à la ligne concernée, quelles que soient les tailles de police contenues dans les colonnes. Si une police est trop grande, le texte sera tronqué. De même, les images seront tronquées ou redimensionnées en fonction de leur format.
- Lorsque vous utilisez l'unité "Ligne(s)", la hauteur est calculée en tenant compte de la taille de police de la ligne concernée. Si plusieurs tailles sont définies, 4D utilisera la plus grande.
  Par exemple, si la ligne contient du "Verdana 18", du "Geneva 12" et du "Arial 9", 4D utilisera "Verdana 18" pour déterminer la hauteur d'une ligne (par exemple 25 pixels). Cette hauteur sera ensuite multipliée par le nombre de lignes défini.

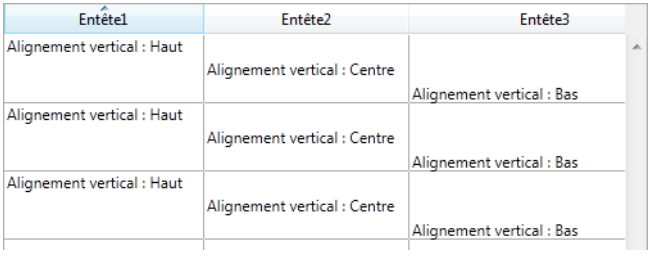
Note Le calcul ne tient pas compte de la taille des images ni des styles éventuellement appliqués aux polices.

- **Conversion des unités** : Lorsque vous basculez d'un type d'unité à l'autre, 4D effectue automatiquement la conversion et affiche le résultat dans la Liste des propriétés. Par exemple, si la police utilisée est "Lucida grande 24", la hauteur "1 ligne" sera convertie en "30 pixels". La hauteur "60 pixels" sera convertie en "2 lignes".

Des conversions multiples peuvent amener à un résultat final différent de la valeur de départ du fait des calculs effectués automatiquement par 4D. Ce principe est illustré par les séquences suivantes :
(*police Arial 18*) : 52 pixels -> 2 lignes -> 40 pixels
(*police Arial 12*) : 3 pixels -> 0,4 ligne arrondi à 1 ligne -> 19 pixels

Alignement vertical des cellules

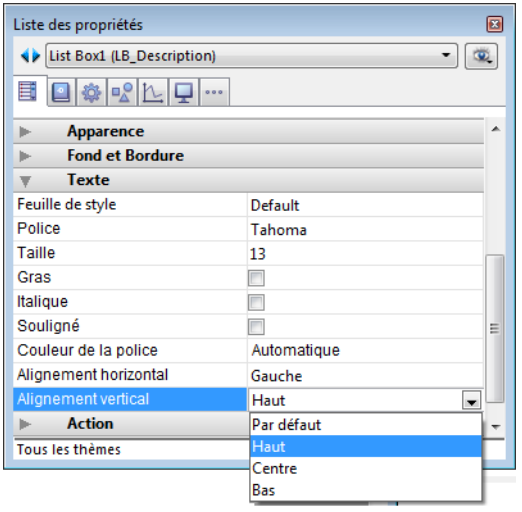
Il est désormais possible de configurer l'alignement vertical des list box.



Cette nouvelle fonction est accessible de deux manières :

- En mode Développement, via la Liste des propriétés (cf. ci-dessous).
- Par programmation, via la nouvelle commande OBJET FIXER ALIGNEMENT VERTICAL, page 175.

Pour définir l'alignement vertical de la list box ou d'un de ses éléments, il suffit de sélectionner une valeur dans le menu **Alignement vertical** du thème Texte :



Le paramétrage peut être appliqué globalement à la list box ou séparément pour chaque colonne. Il est également disponible pour les zones d'en-têtes et de pied.

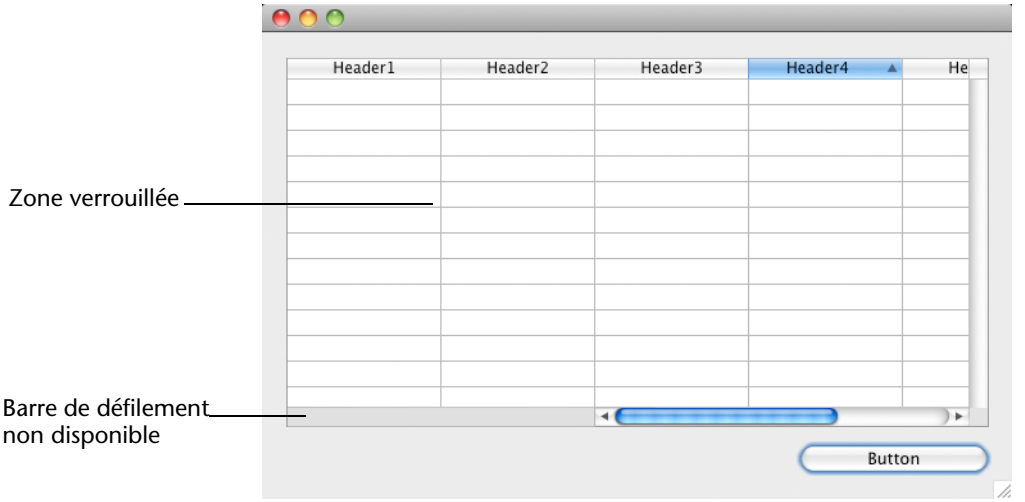
L'option **Par défaut** correspond à :

- **Bas** pour tous les types de données sauf images
- **Haut** pour les données de type image.

Colonnes verrouillées

Avec 4D v13, il est possible de verrouiller une ou plusieurs colonne(s) de list box.

Une colonne verrouillée reste affichée en permanence dans la partie gauche de la list box, même lorsque l'utilisateur fait défiler horizontalement les colonnes. Dans une list box contenant des colonnes verrouillées, la barre de défilement horizontale n'est pas disponible sous ces colonnes :



Une colonne verrouillée peut être redimensionnée, saisissable, etc. comme toute colonne. Seule sa position dans la list box est verrouillée (pas de défilement).

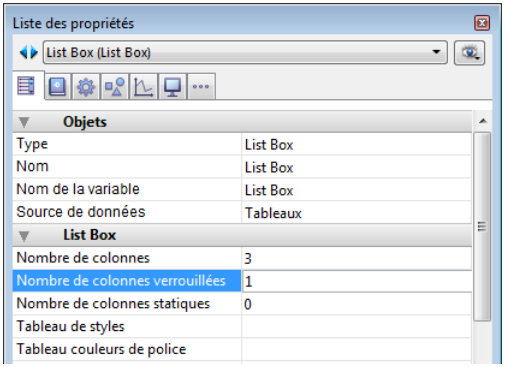
Ce principe est utile par exemple pour afficher des "titres" de lignes dans un tableau de grande dimension.

Définir des colonnes verrouillées

Vous pouvez définir des colonnes verrouillées de deux manières :

- en mode Développement, via la Liste des propriétés,
- par programmation, via la nouvelle commande LISTBOX FIXER COLONNES VERROUILLEES, page 151.

Pour définir des colonnes verrouillées dans l'éditeur de formulaires, sélectionnez la list box et saisissez un nombre de colonnes dans l'option **Nombre de colonnes verrouillées** :



Note Cette option vous permet de définir une "zone de verrouillage". Si une colonne verrouillée est supprimée par programmation, le nombre de colonnes verrouillées dans la list box est diminué de 1. De même, si une colonne est insérée par programmation dans la zone de verrouillage, elle est automatiquement verrouillée.

Colonnes verrouillées et colonnes statiques

Les colonnes verrouillées et les colonnes statiques sont deux fonctionnalités distinctes et indépendantes dans les list box :

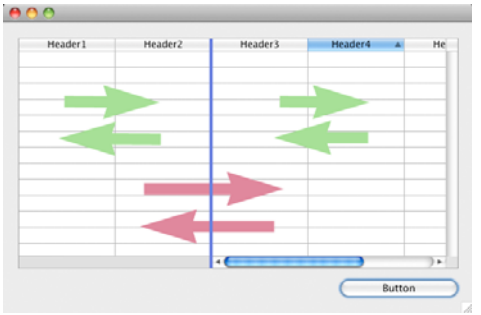
- les colonnes verrouillées restent toujours affichées à gauche de la list box, elles ne défilent pas horizontalement.
- les colonnes statiques ne sont pas déplaçables par glisser-déposer à l'intérieur de la list box.

Notes - Colonnes statiques" est le nouveau nom de la propriété existante "Colonnes fixes" des versions précédentes de 4D.
- Dans 4D v13, vous pouvez définir des colonnes statiques par programmation à l'aide de la nouvelle commande LISTBOX FIXER COLONNES STATIQUES, page 151.

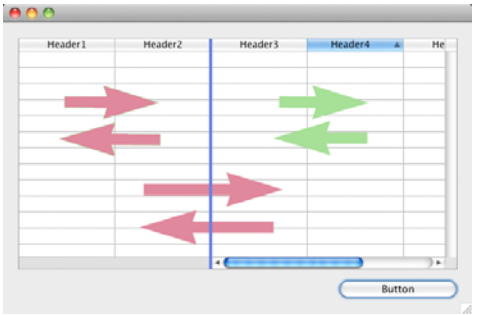
Ces propriétés interagissent de la manière suivante :

- Si vous définissez uniquement des colonnes statiques, elles ne peuvent pas être déplacées (comme dans la version précédente de 4D).
- Si vous définissez des colonnes verrouillées mais non statiques, vous pouvez intervertir librement leur position à l'intérieur de la zone

verrouillée. En revanche, une colonne verrouillée ne peut pas être déplacée à l'extérieur de cette zone.



- Si vous définissez un même nombre de colonnes verrouillées et de colonnes statiques, il ne sera pas possible de déplacer de colonne à l'intérieur de la zone verrouillée.



- Vous pouvez définir toute combinaison de nombre de colonnes verrouillées et de colonnes statiques en fonction de vos besoins. Par exemple, si vous définissez trois colonnes verrouillées et une colonne statique, l'utilisateur pourra intervertir les deux colonnes les plus à droite de la zone verrouillée (puisque la première est statique).

Nouveaux événements pour les list box hiérarchiques

La prise en charge de deux nouveaux événements formulaire permet d'optimiser l'affichage et la gestion des list box hiérarchiques : **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.

La prise en charge des événements formulaire **Sur déployer** et **Sur contracter** permet de s'affranchir de ces contraintes : il est désormais 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 :

Tableau généré - le tableau de la deuxième colonne est généré avec une seule valeur afin que les boutons déployer/contracter soient affichés

Bouton 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.

Insertion de lignes dans la listbox et alimentation des tableaux Artistes, Albums...

- 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.

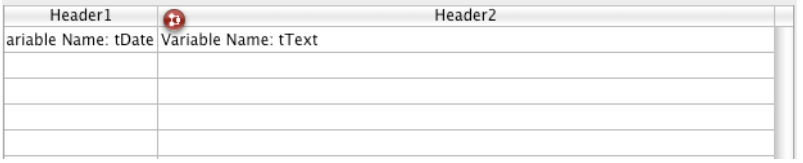
Gestion de la saisie dans les cellules

La prise en charge de la saisie dans les cellules de list box a été modifiée dans 4D v13, pour tous les types de list box (tableau et sélection).

Pour qu'une cellule de list box soit saisissable, il est désormais 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 **Sur avant saisie**, $0 ne retourne pas -1. Désormais, 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.



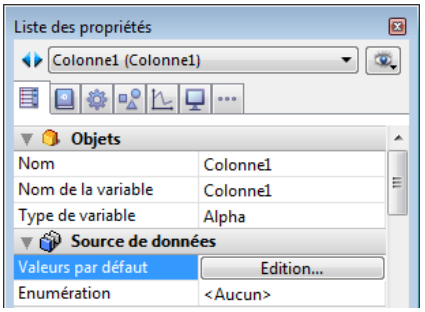
Voici la méthode de la colonne *tText* :

```
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 de compatibilité Avec 4D v13, l'événement **Sur avant saisie** est retourné avant **Sur gain focus**, ce qui est le contraire des versions précédentes de 4D. Cette différence de fonctionnement pourra nécessiter quelques ajustements pour les cellules de texte saisissables : si auparavant le texte saisi était disponible dans l'événement **Sur avant saisie** et pouvait être obtenu via la commande Lire texte edite, dans 4D v13, cette commande retournera une chaîne vide dans ce contexte. Il est nécessaire de déplacer le code de traitement de la saisie dans l'événement **Sur gain focus**.

Valeurs par défaut

Vous pouvez désormais affecter une liste de valeurs par défaut aux colonnes de list box. Pour cela, il suffit de cliquer sur le bouton **Edition...** de la ligne **Valeurs par défaut** de la liste des propriétés pour la colonne :



Vous pouvez ensuite saisir une liste de valeurs séparées par des retours chariot dans la boîte de dialogue d'édition standard. Ces valeurs seront automatiquement accessibles dans la variable tableau associée à la colonne lors de l'exécution du formulaire.

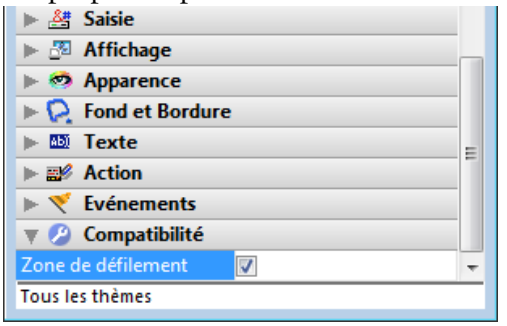
Compatibilité Zones de défilement

Dans les bases de données converties en v13, les anciennes "zones de défilement" sont automatiquement transformées en list box (cf. paragraphe "Zone de défilement", page 16). Afin de reproduire l'interface d'origine, ces list box fonctionnent dans un mode spécifique activé via deux options de l'éditeur de formulaires :

- l'option **Zone de défilement** du thème "Compatibilité",
- la **connexion** des list box si vous utilisiez un ensemble de **zones de défilement groupées**.

Option "Zone de défilement"

L'option **Zone de défilement** est automatiquement cochée dans la Liste des propriétés pour les list box issues de zones de défilement :



Lorsque cette option est cochée pour une list box, les fonctionnements spécifiques suivants sont mis en oeuvre :

- Si le tableau (unique) de la list box a la propriété "invisible", l'objet list box est également entièrement invisible.
- Affecter une valeur au tableau sélectionne la ligne dans la list box (ex : MyArray:=5 sélectionne la 5e ligne de la list box).
- Inversement, cliquer sur une ligne de l'objet modifie la valeur courante du tableau.
- Lorsqu'un déposer est effectué depuis une ligne de la list box sur un objet externe, la commande PROPRIETES GLISSER DEPOSER exécutée dans cet objet retourne un pointeur sur le tableau de la list box (et non la list box elle-même).

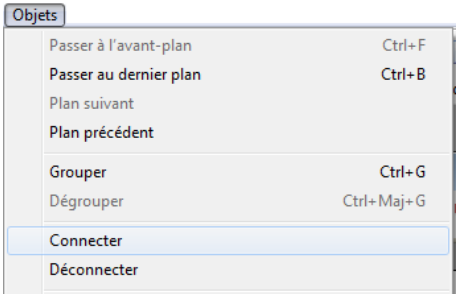
List box connectées

Les list box issues de la conversion d'anciennes zones de défilement groupées sont **connectées**. Les list box connectées ont un fonctionnement coordonné :

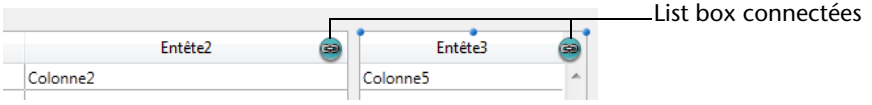
- la sélection d'une ligne dans une list box entraîne la sélection de la même ligne dans les list box appartenant au même groupe de connexion,
- le défilement vertical d'une list box entraîne un défilement identique dans les list box appartenant au même groupe de connexion.

Note Les list box converties sont également **groupées** dans le formulaire (fonction standard de 4D).

La connexion et la déconnexion des list box sont gérées via les nouvelles commandes **Connecter** et **Déconnecter** placées dans le menu **Objets** de l'éditeur de formulaires :



Ces commandes sont activées contextuellement, lorsque plusieurs list box sont sélectionnées dans le formulaire. Lorsqu'une list box connectée (i.e. appartenant à un groupe de connexion) est sélectionnée, un "badge" spécifique est affiché sur toutes les list box appartenant au même groupe de connexion :



Note de compatibilité

Ces principes permettent de reproduire le fonctionnement des zones de défilement groupées. Il est toutefois conseillé d'adapter les formulaires convertis en exploitant les fonctionnalités standard des list box.

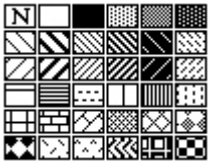
Mise à jour de l'éditeur de formulaires

L'éditeur de formulaires de 4D v13 a été mis à jour :

- suppression des motifs de fond et de bordure avec conversion automatique des motifs existants,
- suppression des commandes obsolètes du menu **Objets**

Conversion des anciens motifs

La prise en charge des motifs de fond et de bordure est supprimée de 4D v13. Conçus à l'origine pour les interfaces monochromes, ces motifs sont désormais obsolètes :

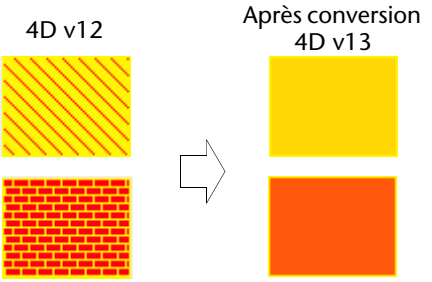


Dans l'éditeur de formulaires de 4D, les commandes de menu et de la Liste des propriétés permettant de définir des motifs ont été supprimées (**Fond**, **Bordure**, **Motif de fond**, **Motif du trait**).

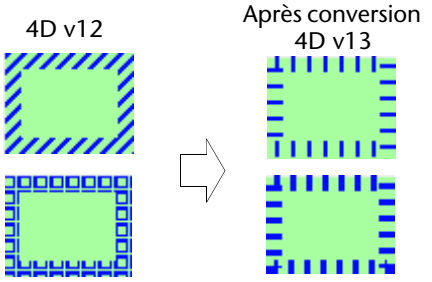
Dans les bases de données converties, les objets qui utilisaient des motifs de fond ou de trait sont automatiquement modifiés de manière à préserver au maximum le rendu initial du formulaire. Les principes mis en oeuvre sont décrits ci-dessous :

- Les motifs de fond des textes (champs, variables ou textes statiques) sont supprimés.
  Si le motif utilisé était "noir" (3e valeur de la palette), il est supprimé et la combinaison couleur de fond/couleur de police est inversée.
- Les motifs de fond des objets graphiques (rectangles, ovales, etc.) sont remplacés par une couleur d'arrière-plan. Cette couleur est obtenue en combinant les précédentes propriétés "Couleur de trait" et "Couleur du fond" de l'objet utilisées pour le motif. Pour définir la part d'importance de chaque couleur dans la combinaison, 4D utilise des coefficients spécifiques en fonction du motif d'origine.

Ce principe donne par exemple les résultats suivants avec un rectangle :



- Les motifs de trait des objets graphiques sont remplacés par des styles de ligne. Le style appliqué automatiquement dépend du motif. Ce principe donne par exemple les résultats suivants :



Mise à jour du menu Objets

Le contenu du menu **Objets** a été modifié afin d'en supprimer les commandes obsolètes ou redondantes. Les commandes suivantes ont été remplacées ou supprimées :

Commandes supprimées du menu Objets dans 4D v12	**Equivalent 4D v13**
Trait	Champ "Epaisseur du trait" de la Liste des propriétés
Fond	Supprimée (cf. paragraphe précédent)
Bordure	Supprimée (cf. paragraphe précédent)
Couleur	- Champ "Couleur" de la Liste des propriétés - Menu contextuel **Couleur**
Police	Champ "Police" de la Liste des propriétés
Style	Champs du thème "Texte" de la Liste des propriétés

Choix du moteur de rendu Web

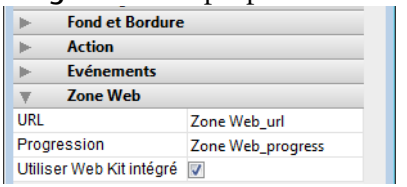
4D v13 vous permet de choisir entre deux nouvelles options pour le rendu des zones Web dans les formulaires : utiliser le "meilleur" moteur de rendu du système ou utiliser le moteur WebKit embarqué.

- **Utiliser le meilleur moteur du système (option sélectionnée par défaut)**
  Cette option équivaut au fonctionnement des versions précédentes de 4D. A noter que sous Windows, 4D v13 utilise désormais automatiquement la plus récente version du navigateur Internet Explorer présent sur la machine (IE9, IE10, etc.). Dans les versions précédentes de 4D, le moteur de IE7 était toujours utilisé. Sous Mac OS, 4D utilise la version courante du WebKit.
  Ce principe vous permet de bénéficier automatiquement des dernières avancées en matière de rendu Web, via HTML5 ou JavaScript. En revanche, vous pouvez rencontrer des différences de rendu entre les implémentation d'Internet Explorer et de WebKit.

- **Utiliser le moteur WebKit intégré**
  4D v13 embarque désormais WebKit, un moteur de rendu Web *open source* et multi-plate-forme. WebKit est notamment utilisé par les navigateurs Safari et Google Chrome.
  L'utilisation du WebKit embarqué vous permet d'avoir l'assurance que le rendu et le fonctionnement des zones Web de votre application seront quasiment identiques, quelle que soit la plate-forme d'exécution de 4D (de légères variations de pixels ou des différences liées à l'implémentation réseau pourront toutefois être constatées).
  En contrepartie, vous ne bénéficiez plus des mises à jour automatiques du moteur Web effectuées par le système d'exploitation. Des nouvelles versions du WebKit seront toutefois proposées via 4D.

Note Le moteur WebKit intégré n'est pas pris en charge par 4D v13 sous Windows XP. Dans ce contexte, le mode standard (meilleur moteur) est toujours activé.

4D v13 vous permet de choisir entre ces deux modes pour chaque zone Web, en fonction des spécificités de votre application. Pour cela, il vous suffit de cocher ou non la nouvelle option **Utiliser Web Kit intégré** dans les propriétés des zones Web :



Par défaut, l'option n'est pas cochée (la zone Web utilise le moteur de rendu système).

Note Lorsque le moteur de rendu Web Kit intégré est utilisé, les fonctionnements suivants sont à noter :
- en cas d'affichage de pages via le protocole HTTPS, l'autorité du certificat n'est pas vérifiée.
- Sous Windows, les fichiers PDF ne peuvent être affichés que dans une fenêtre externe.

Nouveau moteur de rendu Direct2D sous Windows

4D v13 utilise sous Windows le nouveau moteur de rendu graphique Direct2D pour l'affichage de la quasi-totalité des objets de l'interface des applications 4D (contrôles, graphiques, textes...), en mode Développement comme en mode Application.

Présentation

Direct2D est une API de rendu graphique vectoriel dédiée à l'affichage des formes graphiques simples ou complexes (ellipses, rectangles, chemins avec courbes de bezier...), des images et des textes.

Comme Direct2D s'appuie sur Direct3D, il profite directement de l'accélération matérielle des cartes graphiques pour le rendu des formes, images et textes. Direct2D peut toutefois basculer sur une implémentation "logicielle" en cas d'absence de support matériel. Dans tous les cas, les performances sont supérieures à celles obtenues avec le précédent moteur de rendu utilisé par 4D sous Windows (GDI/GDIPlus).

Utilisation et contrôle dans 4D

Direct2D n'est disponible que sous Windows et seulement à partir de Vista.

Dans 4D v13, son activation dépend de la version de Windows et des ressources disponibles sur la machine :

- sous Windows 7 et les versions ultérieures, Direct2D est activé par défaut et tire parti de l'accélération matérielle, lorsqu'elle est disponible, pour les éditeurs de structure, de code et le rendu du SVG. Si elle n'est pas disponible et dans le cas des autres éditeurs, 4D utilise l'émulation "logicielle" (D2D WARP).
- sous Windows Vista, Direct2D est activé par défaut si l'accélération matérielle est disponible pour les éditeurs de structure, de code et le rendu du SVG. Dans les autres contextes ou si l'accélération matérielle n'est pas disponible, 4D utilise GDI/GDIPlus. Ce fonctionnement est transparent pour l'utilisateur.

Note Certains contrôles d'ancienne génération ne sont pas compatibles Direct2D, il restent dessinés avec l'ancien moteur de rendu.

Vous pouvez contrôler l'activation du moteur Direct2D dans votre application à l'aide de nouveaux sélecteurs, accessibles via les commandes FIXER PARAMETRE BASE, Lire parametre base (cf. page 129).

Aperçu avant impression XPS sous Windows

Sous Windows 4D v13 génère désormais les aperçus avant impression sous forme de documents XPS. Le format XPS (*XML Paper Specification*) est une spécification ouverte d'un langage de description de page, développé à l'origine par Microsoft puis standardisé par Ecma International. Aujourd'hui largement répandu parmi les applications Windows, il permet une représentation fidèle des documents imprimés à l'écran et est indépendant de la configuration matérielle.

Note Sous Mac OS, 4D utilise le format PDF, nativement implanté dans le système, pour générer les aperçus.

Prise en charge par le système d'exploitation

Pour générer les documents XPS d'aperçu, 4D fait appel au pilote XPS. Ce pilote est intégré dans Windows à compter de Windows Vista mais peut être obtenu sous Windows XP en installant le framework .NET.

La visualisation s'effectue dans un lecteur XPS, disponible nativement ou via Internet Explorer, en fonction de la version de Windows.

Le tableau suivant résume la disponibilité du pilote et du lecteur XPS suivant les versions de Windows :

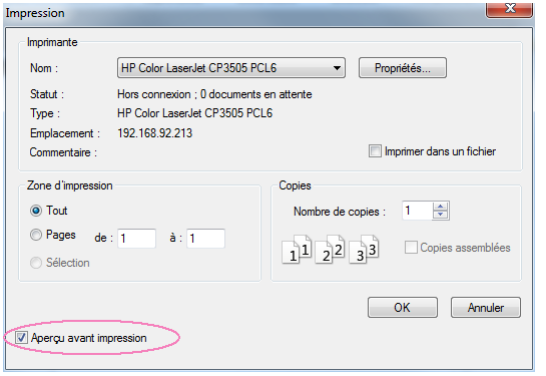
Versions de Windows	Pilote XPS	Lecteur XPS
Windows XP	non	non
Windows XP + framework .NET 3.0	oui	oui via IE
Windows Vista	oui	oui via IE(*)
Windows 7 et suivants	oui	oui (*reader.exe*)

(*) Le lecteur est également disponible séparément.

Fonctionnement dans 4D

Dans 4D v13, la prévisualisation via XPS est utilisée pour :

- l'ensemble des formulaires et des objets 4D, en mode Développement et Application,
- les éditeurs d'étiquettes et d'états rapides.



Lorsqu'une prévisualisation d'impression est générée, 4D redirige l'impression vers un fichier XPS temporaire et lance l'application associée avec les fichiers .XPS.

Note Si cette application n'est pas le visualisateur XPS (c'est le cas par exemple avec Google Chrome), 4D force l'exécution d'Internet Explorer.

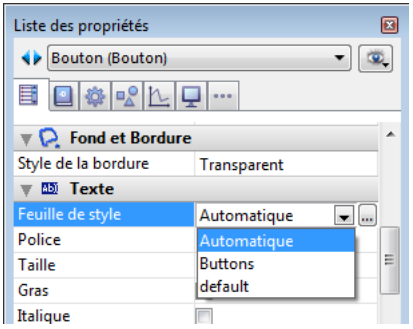
Les fichiers XPS temporaires sont stockés dans le dossier local de l'utilisateur : *...:\Users\{nomUtilisateur}\AppData\Local\Temp\4D\PrintPreview\FolderNumber\NumUUID*

Pour ne pas risquer de saturer le volume, seuls les 100 derniers documents XPS sont conservés. Si l'utilisateur ne dispose pas des droits d'accès adéquats, l'erreur -9799 est retournée. A noter que Windows peut vider le dossier "Temp" au démarrage.
L'ouverture automatique du document par le lecteur ne permet pas de le nommer immédiatement ; c'est la raison pour laquelle un sous-dossier dont le nom est un UUID est créé dans le dossier numéroté afin de contenir le document XPS. Le document est nommé de manière définitive à la fin de l'impression écran. Par défaut, le nom de la fenêtre depuis laquelle l'impression a été lancée est utilisé. Vous pouvez modifier ce nom en appelant la commande CHANGER TITRE FENETRE ou FIXER OPTION IMPRESSION (Option nom document à imprimer; "xx") avant l'impression.

Note Dans 4D v13, deux nouvelles commandes permettent un contrôle renforcé sur l'option d'aperçu (cf. commande Lire apercu impression, page 148 et commande Est en apercu impression, page 148).

Feuille de style Automatique

Une nouvelle feuille de style est disponible dans la Liste de propriétés : **Automatique**.



Note Cette feuille de style est accessible dans la Liste des propriétés uniquement, elle n'apparaît pas dans la liste des feuilles de style de la Boîte à outils.

A la différence des autres feuilles de style, la feuille de style "Automatique" ne possède pas de propriétés prédéfinies mais détermine dynamiquement la **police** et la **taille de police** à utiliser pour l'objet en fonction des paramètres système. Ces paramètres dépendent :

- de la plate-forme,
- de la langue du système,
- du type d'objet de formulaire.

Ce fonctionnement automatique est mis en oeuvre à chaque utilisation du formulaire, en mode Développement ou en mode Application. Avec cette feuille de style, vous avez la garantie que les libellés seront toujours affichés selon les paramètres courants du système. En contrepartie, leurs dimensions pourront varier d'une machine à l'autre.

La feuille de style "Automatique" gère la police et la taille de police. Si vous modifiez une de ces propriétés dans l'éditeur de formulaires, la feuille de style cesse de fonctionner de manière dynamique. En revanche, vous pouvez appliquer des propriétés de style personnalisées (**Gras**, **Italique** ou **Souligné**) en conservant son fonctionnement.

A compter de 4D v13, cette feuille de style est appliquée par défaut à tout nouvel objet créé dans l'éditeur de formulaires.
Bien entendu, les feuilles de style utilisées dans les bases de données converties ne sont pas modifiées. Si vous souhaitez remplacer automatiquement vos feuilles de styles existantes, vous pouvez utiliser la fonction standard de rechercher/remplacer qui prend désormais en charge les références de feuilles de style.

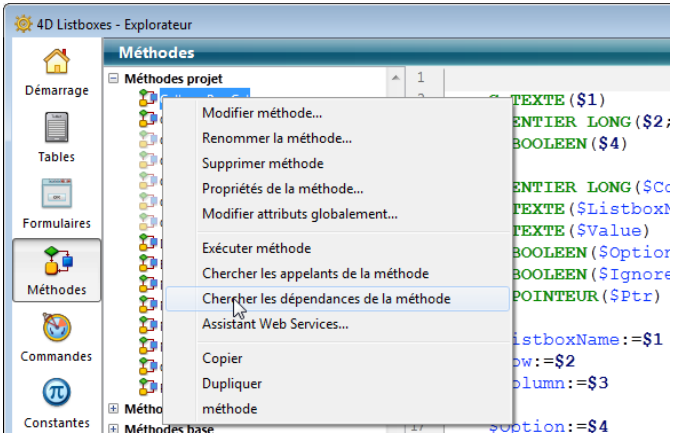
Recherche et déplacement des dépendances

Il est désormais possible des rechercher automatiquement les *dépendances* des méthodes projet, c'est-à-dire les autres méthodes projet et les formulaires utilisés directement ou indirectement par ces méthodes.

Les dépendances trouvées sont affichées dans une fenêtre de résultat de recherche. Il est alors possible de les sélectionner et de les transférer dans une autre base de données via le glisser-déposer. Ce principe facilite le déplacement de fonctionnalités entières entre vos bases.

Lancer la recherche

Pour rechercher les dépendances d'une méthode projet, il suffit de la sélectionner dans l'Explorateur et de choisir la commande **Chercher les dépendances** dans le menu contextuel de la fenêtre :



Les dépendances de la méthode sélectionnée sont alors affichées dans une fenêtre de résultat ; vous pouvez les ouvrir dans leur éditeur ou les sélectionner afin de les déplacer par glisser-déposer.

Vous pouvez également sélectionner plusieurs méthodes projet et choisir la commande **Chercher les dépendances**. Dans ce cas, les dépendances de toutes les méthodes sélectionnées sont affichées dans la fenêtre de résultat.

Analyse des dépendances

Pour trouver les méthodes projet dépendantes, 4D analyse le code de la méthode et détecte :

- les appels directs dans le code,
- les noms de méthodes passés en paramètres à une des commandes suivantes :
  Creer fenetre (paramètre *caseFermeture*)
  APPELER SUR ERREUR
  APPELER SUR EVENEMENT
  Nouveau process
  EXECUTER SUR CLIENT
  FIXER METHODE LIGNE MENU
  EXECUTER METHODE

Le nom passé en paramètre doit être une constante littérale de type texte correspondant à un nom de méthode exécutable valide.

Pour trouver un formulaire dépendant dans du code, 4D détectera un nom de formulaire passé en paramètre à une des commandes suivantes :

- Imprimer ligne
- DIALOGUE
- FORM FIXER SORTIE
- FORM FIXER ENTREE
- PARAMETRES IMPRESSION
- FORM LIRE PROPRIETES
- Creer fenetre formulaire

Le paramètre *table* du formulaire sera reconnu mais doit être précisé explicitement.

Cas de non-détection

4D ne pourra pas détecter de dépendance dans les cas suivants :

```
  //méthode par référence
v:="monhandler"
APPELER SUR ERREUR( v )

  //pointeur vers une table
DIALOGUE( Table(1)-> ; $nomform)

  // utilisation de TABLE PAR DEFAUT
TABLE PAR DEFAUT([maTable])
DIALOGUE( "monForm")
  // "monForm" sera évalué comme un formulaire projet

  // appel de fonction via FN dans le code SQL
Debut SQL
  SELECT Titre_Film, {FN Quel_Nb_Acteurs(ID) AS NUMERIC}
  ...
Fin SQL
```

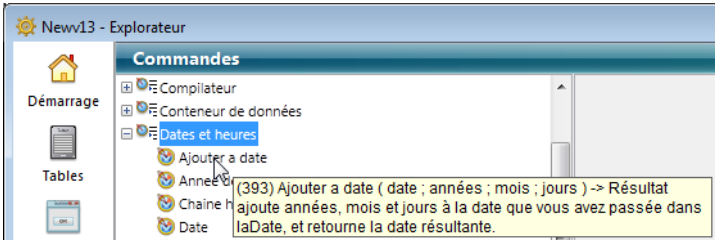
Glisser déposer depuis la fenêtre de résultat

Vous pouvez glisser-déposer des éléments depuis la fenêtre de résultat afin de déplacer des objets entre deux applications en mode Développement. Les principes de déplacement mis en oeuvre (notamment les "objets indissociables") sont identiques à ceux des déplacements depuis l'Explorateur (cf. manuel *Mode Développement*).

Dans 4D v13, les tables référencées dans les méthodes sont désormais déplacées par défaut avec les méthodes. Vous pouvez désactiver ce fonctionnement en appuyant sur la touche **Maj** pendant le déplacement depuis la fenêtre de résultat ou l'Explorateur.

Infobulles dans l'Explorateur

La zone de liste de l'Explorateur affiche désormais le numéro et la syntaxe des commandes ainsi que la valeur des constantes sous forme d'infobulles :

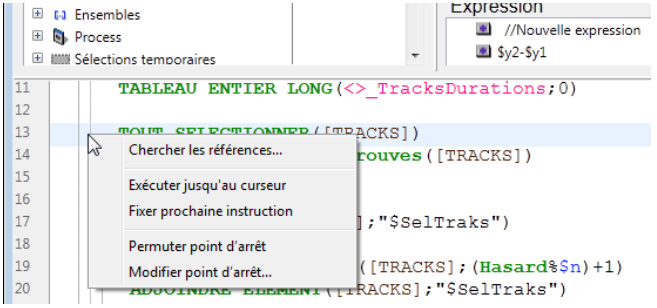


Ces infobulles sont identiques à celles qui sont affichées dans la zone d'édition de l'éditeur de méthodes. Dans 4D v13, elles sont également affichées dans les zones de liste de l'éditeur de méthodes.

Menu contexuel du débogueur

L'interface du débogueur de 4D v13 a été harmonisée avec celle de l'éditeur de méthodes : en particulier, les numéros de ligne sont désormais affichés et vous avez la possibilité de contracter/déployer le code.

Vous disposez également d'un nouveau menu contextuel donnant accès à des 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).

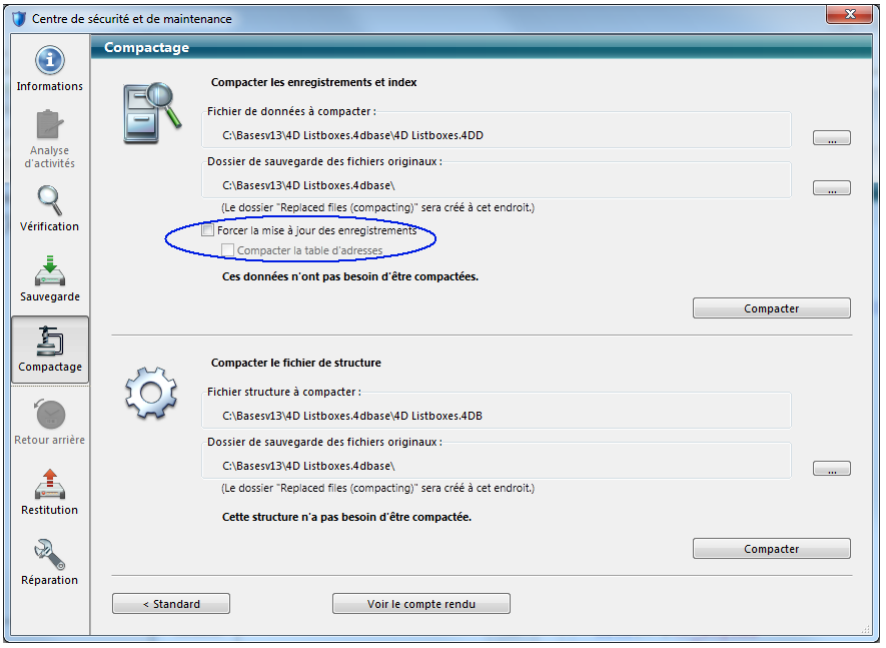
Note de compatibilité Cette fonction remplace les "points d'arrêt provisoires", supprimés à compter de 4D v13.

- **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.

Centre de Sécurité et de Maintenance

Nouvelles options de compactage

Deux nouvelles options sont disponibles dans la page **Compactage/Avancé** pour le compactage des enregistrements et index (données) :



Ces options sont également disponibles dans la commande Compacter fichier donnees, page 128.

- **Forcer la mise à jour des enregistrements** : lorsque cette option est cochée, 4D réécrit chaque enregistrement de chaque table lors de l'opération de compactage, en fonction de sa description en structure. Lorsque l'option n'est pas cochée, 4D réorganise uniquement le stockage des données sur le disque.

  Cette option est utile dans les cas suivants :

  - lorsque des modifications de types de champs ont été apportées à la structure d'une application après que des données ont été saisies. Par exemple, vous pouvez avoir changé le type d'un champ Entier long en Réel. 4D autorise même des modifications entre des types très différents (avec risques de pertes de données), par exemple un champ Réel peut être changé en Texte et inversement.

Dans ce cas, 4D ne convertit pas rétroactivement les données déjà saisies, elles ne sont converties que si les enregistrements sont chargés puis sauvegardés. L'option permet de forcer la conversion de toutes les données.

- lorsqu'une option de stockage externe des données de type Texte, Image ou BLOB a été modifiée après que des données aient été saisies. Ce cas peut notamment se produire après conversion d'une base en v13 car de nouvelles options sont disponibles (cf. paragraphe "Stockage externe des données", page 24). Comme pour le cas du retypage ci-dessus, 4D ne modifie pas rétroactivement les données déjà saisies. Pour cela, vous pouvez forcer la mise à jour des enregistrements afin d'appliquer le nouveau mode de stockage aux enregistrements déjà saisis.
- lorsque des champs ou des tables ont été supprimé(e)s. Dans ce cas, le compactage avec mise à jour des enregistrements permet de récupérer la place de ces données supprimées et donc de réduire la taille du fichier.

Note La sélection de cette option entraîne la mise à jour de tous les index

- **Compacter la table d'adresses** (actif uniquement lorsque la précédente option est cochée) : cette option provoque la reconstruction complète de la table d'adresses des enregistrements au moment du compactage. Cette opération permet d'optimiser la taille de la table d'adresses. Elle est principalement utile dans les bases de données où de très nombreux enregistrements ont été créés puis supprimés. Dans les autres cas, l'optimisation ne sera pas déterminante.

  A noter que cette option ralentit le compactage de façon conséquente et qu'elle rend invalides les ensembles sauvegardés via la commande STOCKER ENSEMBLE. Il est d'ailleurs fortement recommandé dans ce cas de supprimer les ensembles sauvegardés car leur utilisation peut conduire à des sélections de données erronées.

Note Le compactage tient compte des enregistrements des tables placées dans la corbeille. La présence d'un grand nombre d'enregistrements dans la corbeille peut constituer un facteur de ralentissement supplémentaire pour l'opération.

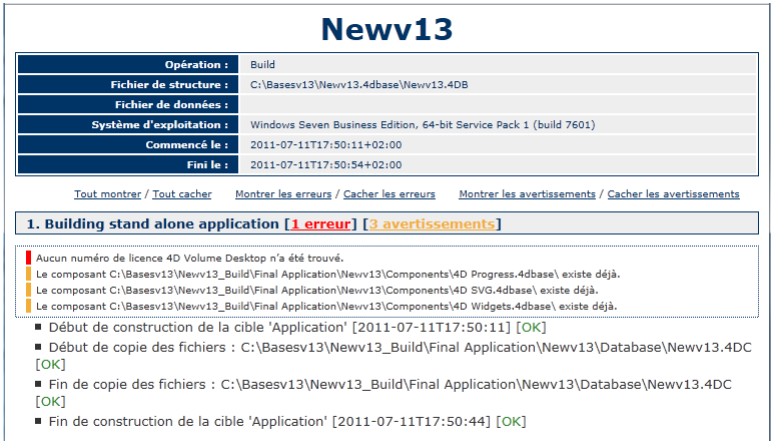
Taille des données externes

La page "Informations/Données" du CSM ne tient pas compte de la taille des données stockées à l'extérieur du fichier de données (cf. paragraphe "Stockage externe des données", page 24).

Log en HTML pour le générateur d'applications

Le générateur d'applications crée désormais un fichier d'historique au format HTML dans le sous-dossier **Logs** de la base, en plus du fichier au format XML disponibles dans les versions précédentes. Ce fichier est nommé BuildApp.log.html. A chaque génération d'application, les deux fichiers sont mis à jour avec les mêmes informations.

Le fichier HTML permet de visualiser les erreurs et avertissements de façon graphique :



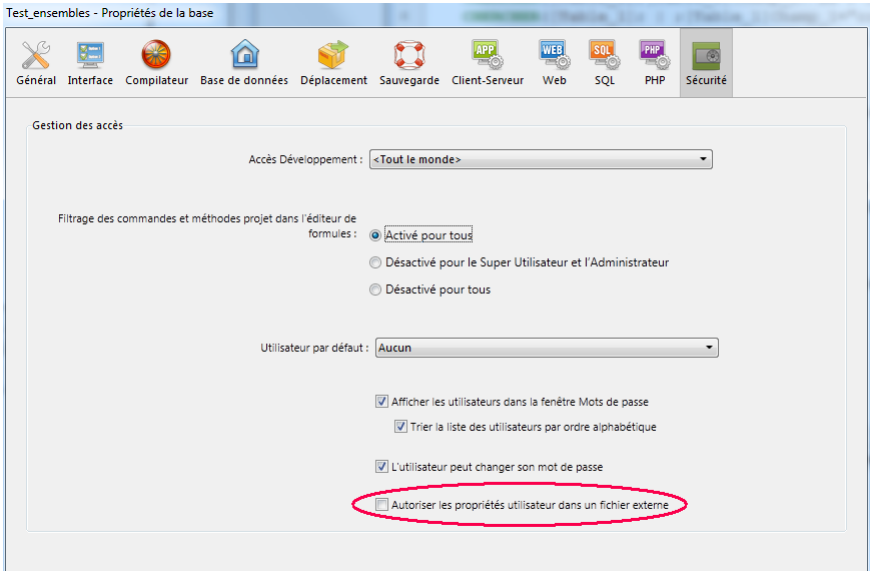
Externaliser des propriétés utilisateur

4D v13 permet de générer un fichier externe contenant des propriétés personnalisées. Lorsque la fonctionnalité est activée, les propriétés contenues dans le fichier externe (appelées "propriétés utilisateur"), lorsqu'elles sont définies, sont utilisées à la place des propriétés stockées dans le fichier de structure de la base (appelées "propriétés structure").

Ce principe permet notamment de conserver des paramétrages personnalisés entre deux mises à jour d'une application 4D, ou encore de pouvoir gérer différents paramétrages pour une même application 4D déployée sur plusieurs sites. Il rend également possible de gérer par programmation des fichiers de propriétés via le XML.

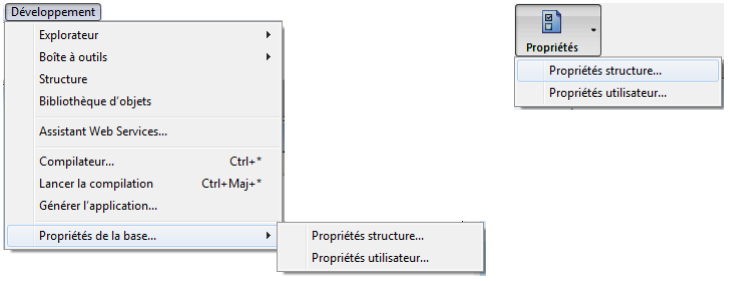
Activer le mode "propriétés utilisateur"

Pour pouvoir utiliser la fonctionnalité d'externalisation des propriétés utilisateur, vous devez cocher l'option **Autoriser les propriétés utilisateur dans un fichier externe**, présente dans la page "Sécurité" des Propriétés de la base :



Note La prise en compte de cette option nécessite le redémarrage de la base.

Lorsque cette option est cochée, les propriétés de la base sont séparées en deux boîtes de dialogue : **Propriétés structure** et **Propriétés utilisateur**. Ces boîtes de dialogue sont accessibles via le menu **Développement/Propriétés de la base>** ou le bouton **Propriétés** de la barre d'outils :

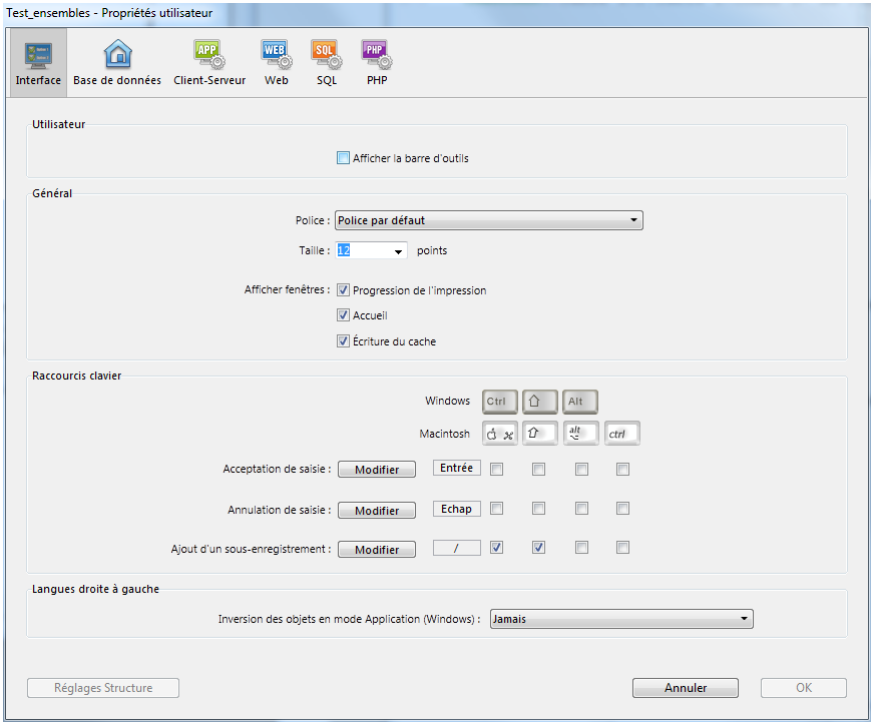


Boîtes de dialogue de gestion des propriétés

Lorsque le mode externe est actif, les propriétés de la base sont accessibles via deux boîtes de dialogue : "Propriétés structure" et "Propriétés utilisateur".

La boîte de dialogue des "Propriétés structure" est identique à celle des Propriétés de la base en mode standard, elle donne accès à toutes les propriétés. La boîte de dialogue des "Propriétés utilisateurs" contient une sélection de propriétés pertinentes qu'il est possible d'externaliser.

Propriétés utilisateur



Conformément au principe de priorité des propriétés utilisateur, si une option est modifiée dans cette boîte de dialogue, sa valeur sera prise en compte à la place de celle stockée dans les propriétés de la structure.

Le tableau suivant liste les pages de propriétés présentes dans la boîte de dialogue "Propriétés utilisateur" et décrit les principales différences avec les propriétés standard :

Pages des Propriétés de la base	Propriétés utilisateur
Général	*Page non disponible*
Interface	Page identique propriétés standard
Compilateur	*Page non disponible*
Base de données	Accès à l'onglet "Mémoire" uniquement
Déplacement	*Page non disponible*
Sauvegarde	*Page non disponible*
Client-Serveur	Page identique propriétés standard
Web	Onglet "Web service" : pas d'option de préfixage des méthodes
SQL	Page identique propriétés standard
PHP	Page identique propriétés standard
Sécurité	*Page non disponible*
Compatibilité	*Page non disponible*

Note Des modifications ont également été apportées à la commande OUVRIR FENETRE PROPRIETES, page 133.

Fichier des propriétés utilisateur

Lorsque l'option **Autoriser les propriétés utilisateur dans un fichier externe** est cochée dans les Propriétés de la base, le fichier de propriétés utilisateur est automatiquement créé et placé à l'emplacement suivant :

[*DossierBase*]/Preferences/**settings.4DSettings**

... où *[DossierBase]* est le nom du dossier contenant le fichier de structure de la base.

Le fichier de propriétés utilisateur, au format XML, peut être lu et modifié à l'aide des commandes XML intégrées de 4D ou via un éditeur XML. Ce principe permet de gérer des propriétés par programmation, notamment dans le contexte d'applications compilées et fusionnées avec 4D Volume Desktop. A noter que si vous modifiez ce fichier par programmation, les modifications ne seront prises en compte qu'à la réouverture de la base.

4 Serveur Web

Le serveur Web inclus dans 4D v13 a été entièrement réécrit afin de bénéficier des technologies les plus récentes et de s'adapter aux évolutions du monde du Web.

Ce nouveau moteur reste entièrement compatible avec les applications Web 4D existantes : le code et les balises HTML, les URLs spéciaux et, de manière générale, tous les mécanismes spécifiques permettant à une application 4D de servir des pages Web statiques ou dynamiques fonctionneront de manière identique en v13. Des ajustements pourront être nécessaires, ils sont détaillés dans ce chapitre. En outre, différentes nouveautés sont disponibles, vous permettant d'enrichir vos applications Web.

Seule exception : le mode contextuel. Déclaré obsolète depuis 4D v12, ce mode n'est plus pris en charge à compter de 4D v13. Si vos applications utilisaient cette fonctionnalité, vous devrez les faire migrer vers des mécanismes standard tels que les sessions Web (nouveauté de 4D v13).

Note Pour plus de clarté, toutes les commandes du thème "Serveur Web" ont été préfixées "WEB" (cf. paragraphe "Renommage des commandes du thème Serveur Web", page 204).

Gestion des sessions Web

Le serveur Web de 4D v13 propose un nouveau mécanisme simple et complet de gestion des sessions utilisateur.

Description

Le mécanisme de gestion automatique des sessions 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 (s'il a expiré), 4D crée un nouveau contexte.

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 nouvelle 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 nouvelle méthode base Sur fermeture session 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.

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 nouvelle commande WEB FIXER OPTION).
Lorsqu'un contexte est sur le point d'être détruit (fermeture du process Web associé), la nouvelle méthode base Sur fermeture session 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.

- 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;pre-
fValues)
   www_UserName:=[User]Name
   www_UserMail:=[User]mail

         // Fin d'initialisation de la session
Fin de si
```

- Code de la méthode base Sur fermeture session Web :

```
   // Sur fermeture session Web
   // Après une période d'inactivité ou en cas de besoin, 4D ferme la session
C_ENTIER LONG(www_SessionID)
www_SessionID:=0
   // 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.
```

Activation et inactivation du mécanisme

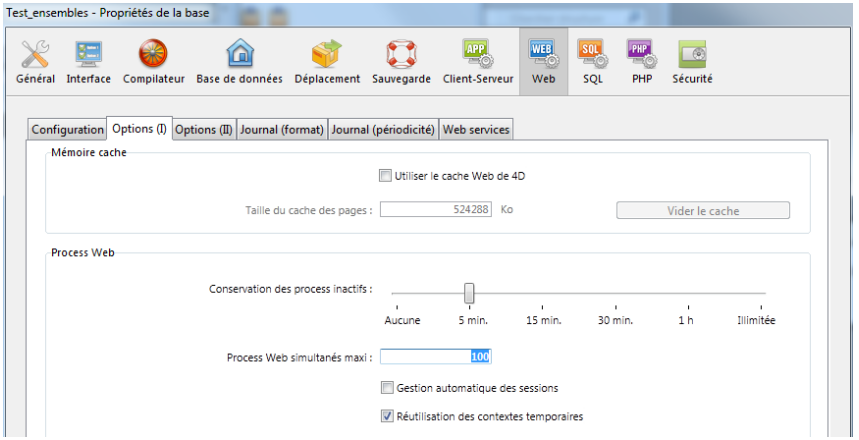
Le nouveau mécanisme de gestion des sessions doit être activé sur votre serveur Web 4D pour que vous puissiez en bénéficier dans votre application.

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 la nouvelle 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 nouvelle commande WEB FIXER OPTION, page 198. 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.

Option "Réutilisation des contextes temporaires (en mode distant)"

L'option existante **Réutilisation des contextes temporaires (en mode distant)** est désormais automatiquement cochée (et verrouillée) lorsque la nouvelle 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" : une fois la session terminée, le process est automatiquement tué (il n'est pas réutilisé).

Comme dans les versions précédentes de 4D, 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 tués après utilisation.

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. 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.

Nouvelles balises

Deux nouvelles balises HTML spéciales sont interprétées par le serveur Web de 4D v13 : **4DELSEIF** et **4DBASE**.

4DELSEIF

La balise **4DELSEIF** permet de construire des structures conditionnelles de type Au cas ou.

Syntaxe : *<!--#4DIF expression1--> <!--#4DELSEIF expression2-->...<!--#4DELSEIF expressionN--> {<!--#4DELSE expression-->} <!--#4DENDIF -->*

A l'aide de cette balise, vous pouvez tester un nombre illimité de conditions. Seul le code suivant la première condition évaluée à Vrai sera exécuté. Si aucune des conditions n'est vraie, aucune instruction n'est exécutée (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.

Ainsi, le code (4D v12) suivant :

```
<!--#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-->
```

... peut être réécrit à l'aide de 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-->
```

- Insertion de 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-->
```

4DBASE

La nouvelle balise **4DBASE** permet de désigner un répertoire courant, qui sera utilisé par la balise 4DINCLUDE.

Syntaxe : *<!--#4DBASE cheminDossier-->*

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.
Passez le mot-clé WEBFOLDER pour rétablir le chemin par défaut (relatif à la page).
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.

Ainsi, le code (4D v12) 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-->
```

- 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* :

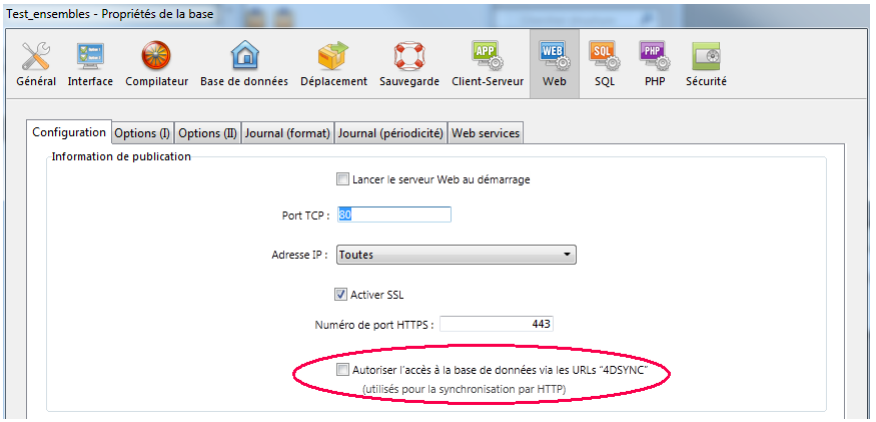
```
/* Head.htm */
/* ici le répertoire courant est relatif au fichier inclus (FR/ ou US/) */
<!--#4DBASE Styles/-->
<!--#4DINCLUDE main.css-->
<!--#4DINCLUDE product.css-->
<!--#4DBASE Scripts/-->
<!--#4DINCLUDE main.js-->
<!--#4DINCLUDE product.js-->
```

Gestion des URLs spéciaux

Option de prise en charge des requêtes /4DSYNC

Dans 4D v13, une nouvelle option 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, reportez-vous à la section http://doc.4d.com/4Dv13/help/Title/fr/page1204.html dans le manuel *Langage* de 4D).

La nouvelle option est nommée **Autoriser l'accès à la base de données via les URLs "4DSYNC"** et est placée dans la page "Web/Configuration" des Propriétés de la base :



Cette nouvelle 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 spécifique (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 v13.

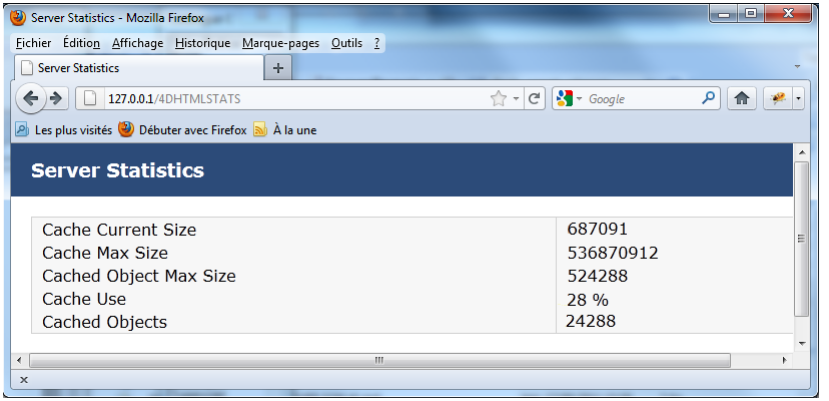
Pour des raisons de compatibilité, cette option est cochée par défaut dans les bases de données converties depuis une version précédente de 4D. Il est recommandé de la désélectionner si votre application n'exploite pas la fonction de réplication par HTTP.

Note Dans les versions précédentes de 4D, il était conseillé de filtrer les URLs /4DSYNC dans la méthode base **Sur authentification Web**. Cette précaution devient inutile dès lors que vous avez pris soin de désélectionner la nouvelle option.

Cette option a une portée locale à l'application et sa prise en compte nécessite un redémarrage du serveur Web.

/4DSTATS et /4DHTMLSTATS

Dans 4D v13, les URLs /4DSTATS et /4DHTMLSTATS renvoient désormais des informations sous la forme d'un tableau HTML :



/4DSTATS

Les informations retournées par l'URL /4DSTATS sont :

- **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.

/4DHTMLSTATS

Comme dans les versions précédentes de 4D, l'URL /4DHTMLSTATS retourne les mêmes informations que /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 désormais le champ **Filtered Objects**.

- **Filtered Objects** : nombre d'objets du cache non comptabilisés par l'URL, notamment les images.

Optimisations

Plusieurs optimisations apportées à 4D v13 permettent d'accélérer l'exécution des applications Web 4D. Ces optimisations ne nécessitent généralement aucune modification du code, mais peuvent entraîner des différences de fonctionnement.

Création différée des process clients sur le serveur

Dans les versions précédentes de 4D, lorsqu'un process global (dont le nom ne commence pas par "$") était créé sur un poste 4D en mode distant, deux process étaient créés immédiatement sur le serveur :

- un process coopératif pour l'accès au langage
- un process préemptif pour les accès au moteur de la base

A partir de 4D v13, chacun de ces deux process n'est créé qu'**en cas de nécessité** lors de l'exécution du code côté client. Ce principe simplifie et optimise les connexions HTTP aux serveurs Web des 4D distants. Il n'est plus nécessaire de choisir entre process global ou process local lors de la création d'un process client pour le serveur Web : il suffit de créer uniquement des process globaux et 4D se charge de n'établir la connexion avec 4D Server qu'en cas de nécessité.

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
$serverTime:=Heure courante(*) // création ici du process coopératif
                                  // sur le serveur
```

Les fonctionnements suivants sont à noter :

- La méthode base Sur ouverture connexion serveur est appelée lors de la création du process coopératif sur le serveur. Cette méthode peut donc être désormais appelée au cours de l'exécution du code côté client, (voire ne pas être appelée) et ce, après ou avant que le process préemptif soit créé.
- Le process principal, c'est-à-dire le premier process créé à la connexion d'un 4D distant, n'est pas touché : les deux process associés sont créés immédiatement sur le serveur.

Compression gzip

Le serveur Web de 4D v13 prend désormais 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, pour un gain de performances immédiat.

Pour que la compression soit utilisée, les conditions suivantes doivent être réunies :

- le client Web accepte la compression (header "Accept-Encoding: gzip")
- le contenu doit être de type texte (les images ou les données binaires ne sont pas compressés). A noter cependant que la commande COMPRESSER BLOB accepte désormais une compression GZIP qui peut être utilisée pour compresser les données envoyées par la commande WEB ENVOYER DONNEES.
- les données doivent être de taille supérieure à 1024 octets (en-dessous de cette taille, le bénéfice de la compression en terme de performances est nul).

Le seuil et le niveau de compression Web peuvent être configurés à l'aide de la nouvelle commande WEB FIXER OPTION, page 198.

5 Langage

Ce chapitre regroupe les nouveautés et modifications apportées au langage de programmation de 4D v13.

Accès objets développement

Dans 4D v13, vous pouvez accéder par programmation au contenu des méthodes de vos applications, en mode interprété. Ce nouveau *source toolkit* facilite l'intégration de vos applications aux outils de contrôle du code, notamment les applications de gestion de versions (*VCS*). Elle 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, le CSM de 4D v13 vous permet de détecter ces doublons.

- L'accès aux objets de l'application 4D s'effectue à l'aide des commandes du nouveau thème "Accès objets développement", par exemple METHODE LIRE NOMS ou METHODE LIRE CHEMINS.
- Les commandes du thème "Accès objets développement" sont utilisables avec 4D en mode local ou distant.

En revanche, gardez à l'esprit que ces commandes ne sont pas utilisables en mode compilé : leur finalité est la création d'outils personnalisés d'aide de développement. Elles 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 modifier les objets (telles que METHODE FIXER ATTRIBUT), car les composants sont toujours exécutés en lecture seulement.

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

Exemple :

- *Form?1* sera encodé *Form%3F1*
- *Button/1* sera encodé *Button%2F1*

Nouvelle balise de macros

Dans le contexte des macro-commandes 4D (éditeur de structure), la nouvelle balise **<method_path>** est disponible : cette balise est remplacée par le chemin d'accès complet de la méthode en cours d'édition.

Chemin methode courante

Chemin methode courante → Texte

Paramètres	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Texte	←	Chemin interne complet de la méthode en cours d'exécution

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.

A noter que la valeur retournée est unique dans la base, à la différence de celle retournée par la commande Nom methode courante.

Note Dans le contexte des macro-commandes 4D, la nouvelle 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} {; *})

Paramètres	Type		Description
laTable	Table	→	Référence de table
tabNoms	Tableau texte	←	Tableau des noms de formulaires
filtre	Texte	→	Filtrage des noms
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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.

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.

- Exemples d'utilisation 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 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 {; *})

Paramètres	Type		Description
chemin	Texte	→	Chemin de méthode projet
typeAttribut	Entier long	→	Type d'attribut
valeurAttribut	Booléen	→	Vrai = sélectionner l'attribut Faux = désélectionner l'attribut
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

La commande METHODE FIXER ATTRIBUT définit 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 définir. Vous pouvez utiliser les constantes suivantes, placées dans le thème "Accès objets développement" :

Constante *(valeur)*	**Description**
Attribut Exécutée sur serveur (8)	Correspond à l'option "Exécuter sur serveur"
Attribut Invisible (1)	Correspond à l'option "Invisible"
Attribut Partagée (5)	Correspond à l'option "Partagée entre composants et base hôte"
Attribut Publiée SOAP (3)	Correspond à l'option "Offerte comme Web Service"
Attribut Publiée SQL (7)	Correspond à l'option "Disponible via SQL"
Attribut Publiée Web (2)	Correspond à l'option "Disponible via les balises HTML et les URLs 4D (4DACTION...)"
Attribut Publiée WSDL (4)	Correspond à l'option "Publiée dans WSDL"

Notes - Pour plus d'informations sur ces attributs, reportez-vous au manuel *Mode Développement* de 4D.
- L'option "Publiée dans WSDL" dépend de l'option "Offerte comme Web Service". Si vous sélectionnez l'attribut Attribut Publiée WSDL sans que l'attribut Attribut Publiée SOAP soit sélectionné, le paramétrage ne sera pas pris en compte.

Passez Vrai dans le paramètre *valeurAttribut* pour sélectionner l'option correspondante et Faux pour la désélectionner.

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.

- Sélection de la propriété "Partagée entre composants et base hôte" pour la méthode projet "Choix dialogue" :

METHODE FIXER ATTRIBUT("Choix dialogue";Attribut Partagée;**Vrai**)

Référence : METHODE Lire attribut

METHODE FIXER CODE

METHODE FIXER CODE (chemin ; code {; *})

Paramètres	Type		Description
chemin	Texte | Tab texte	→	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
code	Texte | Tab texte	→	Code de(s) méthode(s) désignée(s)
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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.

Si la première ligne du *code* d'une méthode contient des métadonnées valides, 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"}

Note Ces métadonnées sont générées automatiquement par la commande METHODE LIRE CODE.

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.

- 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)
```

Référence : METHODE LIRE CODE

METHODE FIXER COMMENTAIRES

METHODE FIXER COMMENTAIRES (chemin ; commentaires {; *})

Paramètres	Type		Description
chemin	Texte | Tab texte	→	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
commentaires	Texte | Tab texte	→	Commentaires de(s) méthode(s) désignée(s)
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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.

- 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)

Référence : METHODE LIRE COMMENTAIRES

METHODE FIXER MODE ACCES

METHODE FIXER MODE ACCES (mode)

Paramètres	Type		Description
mode	Entier long	→	Mode d'accès aux objets verrouillés

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 (valeur)	Description
Sur objet verrouillé abandonner (0)	Le chargement de l'objet est abandonné (fonctionnement par défaut)
Sur objet verrouillé réessayer (1)	4D tente de charger l'objet jusqu'à ce qu'il soit libéré
Sur objet verrouillé confirmer (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)

METHODE Lire attribut

METHODE Lire attribut(chemin ; typeAttribut {; *}) → Booléen

Paramètres	Type		Description
chemin	Texte	→	Chemin de méthode projet
typeAttribut	Entier long	→	Type d'attribut à connaître
*		→	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

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 *(valeur)*	Description
Attribut Exécutée sur serveur (8)	Option "Exécuter sur serveur"
Attribut Invisible (1)	Option "Invisible"
Attribut Partagée (5)	Option "Partagée entre composants et base hôte"
Attribut Publiée SOAP (3)	Option "Offerte comme Web Service"
Attribut Publiée SQL (7)	Option "Disponible via SQL"
Attribut Publiée Web (2)	Option "Disponible via les balises HTML et les URLs 4D (4DACTION...)"
Attribut Publiée WSDL (4)	Option "Publiée dans WSDL"

Note Pour plus d'informations sur ces attributs, reportez-vous au manuel *Mode Développement* de 4D.

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.

Référence : METHODE FIXER ATTRIBUT

METHODE Lire chemin

METHODE Lire chemin (typeMéthode {; laTable}{; nomObjet{; nomObjetForm}} {; *}) → Texte

Paramètres	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
*		→	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

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 *(valeur)*	Description
Chemin Formulaire projet (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 (16)	Chemin des méthodes formulaire table et de toutes leurs méthodes objet. Exemple : [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 (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 (1)	Nom de la méthode. Exemple : MaMethodeProjet
Chemin Trigger (8)	Chemin des triggers de la base. Exemple : [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.

- 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")

Référence : METHODE RESOUDRE CHEMIN

METHODE LIRE CHEMINS

METHODE LIRE CHEMINS({nomDossier;} typeMéthode; tabChemins{; marqueur} {; *})

Paramètres	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	Var Entier long	→	Valeur minimum de marqueur
		←	Nouvelle valeur courante
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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 *(valeur)*	Description
Chemin Formulaire projet (4)	La commande retourne le chemin des méthodes formulaire projet et de toutes leurs méthodes objet. Exemple : [projectForm]/monForm/{formMethod} [projectForm]/monForm/bouton1 [projectForm]/monForm/ma%2liste [projectForm]/monForm2/bouton1
Chemin Formulaire table (16)	La commande retourne le chemin des méthodes formulaire table et de toutes leurs méthodes objet. Exemple : [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 (2)	La commande retourne le 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 (1)	La commande retourne le nom de la méthode. Exemple : MaMethodeProjet
Chemin tous les objets (31)	La commande retourne la combinaison des chemins de toutes les méthodes
Chemin Trigger (8)	La commande retourne le chemin des triggers de la base. Exemple : [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.

- Récupération des méthodes projet placée dans un dossier "web" :

  METHODE LIRE CHEMINS("web";Chemin Méthode projet; tabChemins)

- Récupération des méthodes base et des triggers :

  METHODE LIRE CHEMINS(Chemin Trigger+Chemin Méthode base; tabChemins)

- Récupération des méthodes projet modifiées depuis la dernière sauvegarde :

```
// 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])
```

- Pour un exemple complet, reportez-vous à la description de la commande METHODE FIXER CODE, page 84.

Référence : METHODE LIRE DOSSIERS

METHODE LIRE CHEMINS FORM

METHODE LIRE CHEMINS FORM({laTable;} tabChemins {; filtre}{; marqueur}{; *})

Paramètres	Type		Description
laTable	Table	→	Référence de table
tabChemins	Tableau texte	←	Tableau des chemins et noms des méthodes
filtre	Texte	→	Filtrage des noms
marqueur	Var Entier long	→	Valeur minimum de marqueur
		←	Nouvelle valeur courante
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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 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.

Note La commande ne liste pas les objets des formulaires hérités ni des sous-formulaires.

- 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
```

- Liste des objets du formulaire projet "dial" :

```
METHODE LIRE CHEMINS FORM(tabChemins;"dial")
```

- 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 {; *})

Paramètres	Type		Description
chemin	Texte | Tab texte	→	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
code	Texte | Tab texte	←	Code de(s) méthode(s) désignée(s)
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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 français avec une version française de 4D et en anglais avec toutes les autres versions.
- Une ligne est ajoutée en en-tête du code généré, contenant des métadonnées utilisées lors de l'import du code, par exemple :
  `// %metadata= {invisible:true,lang:"fr"}`
  En cas d'import, cette ligne n'est par importée, seuls les attributs définis sont pris en compte. 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).

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.

▸ Un exemple complet d'exportation et d'importation de méthodes est fourni dans la description de la commande METHODE FIXER CODE, page 84.

Référence : METHODE FIXER CODE

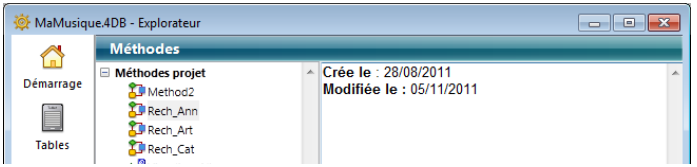
METHODE LIRE COMMENTAIRES

METHODE LIRE COMMENTAIRES (chemin ; commentaire {; *})

Paramètres	Type		Description
chemin	Texte | Tab texte	→	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
commentaires	Texte | Tab texte	←	Commentaires de(s) méthode(s) désignée(s)
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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 commentaires dans le code, qui peuvent être lus 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.

Référence : METHODE FIXER COMMENTAIRES

METHODE LIRE DATE MODIFICATION

METHODE LIRE DATE MODIFICATION(chemin ; dateMod; heureMod{; *})

Paramètres	Type		Description
chemin	Texte | Tab texte	→	Texte ou Tableau texte contenant un ou plusieurs chemin(s) de méthode(s)
dateMod	Date | Tab date	←	Date(s) de modification de méthode(s)
heureMod	Heure | Tab entier long	←	Heure(s) de modification de méthode(s)
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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.

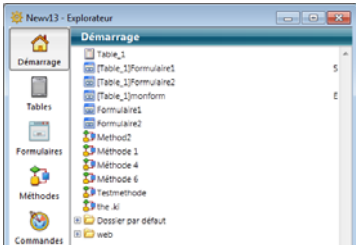
Si la commande est exécutée depuis un composant, elle s'applique par défaut aux méthodes du composant. Si vous passez le paramètre *, elle accède aux méthodes de la base hôte.

METHODE LIRE DOSSIERS

METHODE LIRE DOSSIERS(tabNoms{; filtre}{; *})

Paramètres	Type		Description
tabNoms	Tableau texte	←	Tableau des noms de dossiers de la page Démarrage
filtre	Texte	→	Filtrage des noms
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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} {; *})

Paramètres	Type		Description
tabNoms	Tableau texte	←	Tableau des noms de méthodes projet
filtre	Texte	→	Filtrage des noms
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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.

- Exemples d'utilisation type :

```
  // 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{; *})

Paramètres	Type		Description
chemin	Texte	→	Chemin de la méthode à ouvrir
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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.

Référence : METHODE Lire chemin

METHODE RESOUDRE CHEMIN

METHODE RESOUDRE CHEMIN(chemin; typeMéthode ; ptrTable; nomObjet; nomObjetForm{; *})

Paramètres	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
*		→	Si passé = la commande s'applique à la base hôte lorsqu'elle est exécutée depuis un composant (paramètre ignoré hors de ce contexte)

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* reçoit une valeur indiquant le type de la méthode. Vous pouvez comparer cette valeur aux constantes suivantes du thème "Accès objets développement" :

Constante *(valeur)*	**Description**
Chemin Formulaire projet (4)	Chemin de méthode formulaire projet et de toutes leurs méthodes objet
Chemin Formulaire table (16)	Chemin de méthode formulaire table et de toutes leurs méthodes objet
Chemin Méthode base (2)	Chemin de méthode base (nom anglais)
Chemin Méthode projet (1)	Chemin de méthode projet
Chemin Trigger (8)	Chemin de triggers

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 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.

▸ 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: ""
```

- 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("[tabletForm]/Table1/output%2A1/
            myVar%2A1";$methodType;$tablePtr;$objectName;
            $objectFormName)
// $methodType: 16
// $tablePtr: -> [Table1]
// $objectName: "output*1"
// $objectFormName: "Btn*1"
```

Référence : METHODE Lire chemin

BLOB

COMPRESSER BLOB

COMPRESSER BLOB(blob {; compression})

La commande COMPRESSER BLOB admet deux nouvelles méthodes de compression, accessibles via les constantes suivantes :

Constante *(valeur)*	**Description**
GZIP Méthode de compression compacte (-1)	Compression GZIP la plus compacte
GZIP Méthode de compression rapide (-2)	Compression GZIP la plus rapide

Note Bien entendu, la commande DECOMPRESSER BLOB prend en charge la décompression GZIP.

- 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 ; *)
```

LIRE PROPRIETES BLOB

LIRE PROPRIETES BLOB(blob ; compressé {; tailleDécompressée {;tailleCourante}})

La commande LIRE PROPRIETES BLOB peut retourner deux nouvelles valeurs dans le paramètre *compressé*, correspondant aux méthodes de compression GZIP (cf. COMPRESSER BLOB).

Chaînes de caractères

LIRE MOTS CLES TEXTE

LIRE MOTS CLES TEXTE(texte; tabMotsClés{; *})

Paramètres	Type		Description
texte	Texte	→	Texte original
tabMotsClés	Tableau texte	←	Tableau contenant les mots-clés
*		→	Si passé = mots uniques

La nouvelle 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://www.unicode.org/reports/tr29/#Word_Boundaries

Note A la demande des utilisateurs, un 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".

Note 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 au 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 3 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).

- 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")
```

- 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
```

- 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))
```

Client HTTP

Le nouveau thème "Client HTTP" contient plusieurs commandes permettant à une application 4D de communiquer avec un serveur HTTP. Ces commandes simplifient l'utilisation du protocole HTTP pour les échanges de données, notamment via des serveurs proxy.

HTTP AUTHENTIFIER

HTTP AUTHENTIFIER(nom; motDePasse{; méthodeAuth}{; *})

Paramètres	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

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 *(valeur)*	Description
HTTP Basic (1)	Utiliser la méthode d'authentification BASIC
HTTP Digest (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 réinitialisées après chaque requête. Vous devez donc utiliser la commande HTTP AUTHENTIFIER avant chaque HTTP Request ou HTTP Get. Il est toutefois possible de conserver temporairement ces informations à l'aide d'une option de la commande HTTP FIXER OPTION. Dans ce cas, il n'est pas nécessaire d'exécuter la commande HTTP AUTHENTIFIER avant chaque requête.

- 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(...)
```

Référence : HTTP FIXER OPTION

HTTP Get

HTTP Get (url; réponse {;nomsEnTêtes; valeursEnTêtes} {;*}) → Entier long

Paramètres	Type		Description
url	Texte	→	URL auquel envoyer la requête
réponse	Texte | BLOB | Image	←	Résultat de la requête
nomsEnTêtes	Tableau Texte	→ ←	Tableau contenant les noms d'en-têtes de la requête
valeursEnTêtes	Tableau Texte	→ ←	Tableau contenant les valeurs d'en-têtes de la requête
*	Opérateur	→	Si passé, la connexion est maintenue (keep-alive) Si omis, la connexion est automatiquement refermée
Résultat	Entier long	←	Code de statut HTTP

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 encodé en UTF16
- BLOB : lorsque le résultat est attendu sous forme binaire.
- Image : lorsque le résultat est attendu sous forme d'image.

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 le type de données renvoyé par le serveur ne correspond pas au type de la variable, elle est retournée vide.

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 l'intercepter à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

- 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)
```

- 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)
```

- Récupération d'une vidéo :

```
C_BLOB(vBlob)
$httpResponse:=HTTP Get("http://www.example.com/video.flv";vBlob)
BLOB VERS DOCUMENT(vBlob;"video.flv")
```

Référence : HTTP Request

HTTP Request

HTTP Request (méthodeHTTP; url; contenu; réponse {;nomsEnTêtes; valeursEnTêtes} {;*}) → Entier long

Paramètres	Type		Description
méthodeHTTP	Texte	→	Méthode HTTP pour la requête
url	Texte	→	URL auquel envoyer la requête
contenu	Texte | BLOB | Image	→	Contenu du corps (*body*) de la requête
réponse	Texte | BLOB | Image	←	Résultat de la requête
nomsEnTêtes	Tableau Texte	→ ←	Tableau contenant les noms d'en-têtes de la requête
valeursEnTêtes	Tableau Texte	→ ←	Tableau contenant les valeurs d'en-têtes de la requêtes
*	Opérateur	→	Si passé, la connexion est maintenue (keep-alive) Si omis, la connexion est automatiquement refermée
Résultat	Entier long	←	Code de statut HTTP

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 *(valeur)*	Description
HTTP Méthode DELETE (DELETE)	Voir la RFC 2616
HTTP Méthode GET (GET)	Voir la RFC 2616. Equivaut à utiliser la commande HTTP Get
HTTP Méthode HEAD (HEAD)	Voir la RFC 2616
HTTP Méthode OPTIONS (OPTIONS)	Voir la RFC 2616
HTTP Méthode POST (POST)	Voir la RFC 2616
HTTP Méthode PUT (PUT)	Voir la RFC 2616
HTTP Méthode TRACE (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 ou image. 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).

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 encodé en UTF16 ou sous forme de référence d'arbre XML.
- BLOB : lorsque le résultat est attendu sous forme binaire.
- Image : lorsque le résultat est attendu sous forme d'image.

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 l'intercepter à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

- 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)
```

- 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)
```

Référence : HTTP Get

HTTP FIXER OPTION

HTTP FIXER OPTION(option; valeur)

Paramètres	Type		Description
option	Entier long	→	Code de l'option à fixer
valeur	Entier long	→	Valeur de l'option

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 "HTTP Client" :

Constante *(valeur)*	**Commentaire**
HTTP Timeout (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.).
HTTP Suivre redirection (2)	*valeur* = 0 (ne pas accepter les redirections) ou 1 (accepter les redirections). Valeur par défaut = 1
HTTP Redirections max (3)	*valeur* = nombre maximum de redirections acceptées Valeur par défaut = 2
HTTP Afficher dial auth (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 Effacer infos auth (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 Compression (6)	*valeur* = 0 (ne pas compresser) ou 1 (compresser). Par défaut : 1 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.

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.

Référence : HTTP LIRE OPTION

HTTP LIRE OPTION

HTTP LIRE OPTION (option ; valeur)

Paramètres	Type		Description
option	Entier long	→	Code de l'option à lire
valeur	Entier long	←	Valeur courante de l'option

La commande HTTP LIRE OPTION retourne la valeur courante des options HTTP (options utilisées par le client pour la prochaine requête déclenchée par les commandes HTTP Get ou HTTP Request).

Pour plus d'informations sur les options HTTP et les valeurs retournées, reportez-vous à la description de la commande HTTP FIXER OPTION.

Note Les options définies sont locales au process courant. Dans le cadre d'un composant, elles sont locales au composant en cours d'exécution.

Référence : HTTP FIXER OPTION

Correcteur orthographique

Ce nouveau thème regroupe les commandes relatives à l'activation et à la gestion par programmation du correcteur orthographique intégré à l'application 4D. Les commandes ont été préfixées "SPELL".

Note Dans les versions précédentes de 4D, les deux commandes existantes CHANGER DICTIONNAIRE et CORRECTION ORTHOGRAPHIQUE étaient placées dans le thème "Outils". Dans 4D v13, elles ont été renommées respectivement SPELL FIXER DICTIONNAIRE COURANT et SPELL CORRECTION ORTHOGRAPHIQUE.

Prise en charge des dictionnaires Hunspell

4D v13 prend désormais 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

Note A compter de la version 13 de 4D, les dictionnaires ne sont plus stockés dans le sous-dossier 4D Extensions des applications 4D.

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 (existant dans les versions précédentes de 4D). Les dictionnaires utilisateurs sont stockés au format UTF-8.

SPELL AJOUTER AU DICTIONNAIRE UTILISATEUR

SPELL AJOUTER AU DICTIONNAIRE UTILISATEUR(mots)

Paramètres	Type	Description
mots	Texte | Tab Texte →	Mot ou liste de mots à ajouter au dictionnaire utilisateur

La nouvelle 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.

- 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)
```

Référence : SPELL VERIFIER TEXTE

SPELL CORRECTION ORTHOGRAPHIQUE

Par soucis de cohérence, la commande existante CORRECTION ORTHOGRAPHIQUE a été renommée SPELL CORRECTION ORTHOGRAPHIQUE. Son fonctionnement est inchangé.

SPELL LIRE LISTE DICTIONNAIRES

SPELL LIRE LISTE DICTIONNAIRES(langID; langFichiers; langNoms)

Paramètres	Type		Description
langID	Tab Entier long	←	ID uniques des langues
langFichiers	Tab Texte	←	Noms des fichiers de langue installés
langNoms	Tab Texte	←	Noms locaux des langues

La nouvelle 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 installés sur la machine.

Cette commande retourne les IDs des dictionnaires Hunspell, disponibles à partir de 4D v13 (cf. paragraphe “Prise en charge des dictionnaires Hunspell”, page 115) ainsi que ceux des dictionnaires Cordial, disponibles dans les versions précédentes de 4D.

- *langID* reçoit les numéros d’ID générés automatiquement et utilisables avec la commande SPELL FIXER DICTIONNAIRE COURANT (nouveau nom de la commande CHANGER DICTIONNAIRE).
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 installés sur le poste. Pour les dictionnaires Cordial, un nom normalisé est retourné ("fr_FR" pour le dictionnaire français, "en_GB" pour le dictionnaire anglais, etc.). Pour les dictionnaires Hunspell, le nom des fichiers (sans extension) est retourné.

- *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.
Pour les dictionnaires Hunspell, 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.

- 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 FIXER DICTIONNAIRE COURANT

SPELL FIXER DICTIONNAIRE COURANT*{(dictionnaire)}*

Paramètres	Type		Description
dictionnaire	Entier long | *Texte*	→	ID *ou Nom* du dictionnaire à utiliser *Si omis = rétablir dictionnaire par défaut*

Note Dans les versions précédentes de 4D, cette commande était nommée CHANGER DICTIONNAIRE.

La syntaxe de la commande SPELL FIXER DICTIONNAIRE COURANT a évolué afin de mieux intégrer les dictionnaires Hunspell (cf. paragraphe “Prise en charge des dictionnaires Hunspell”, page 115) :

- le paramètre *dictionnaire* est désormais optionnel : si la commande est appelée sans paramètre, le dictionnaire par défaut est rétabli.
- le paramètre *dictionnaire* peut désormais être de type Texte : vous pouvez passer dans ce paramètre un nom de dictionnaire Hunspell -- correspondant au nom du fichier de dictionnaire avec ou sans extension. La variable *OK* prend la valeur 1 si le dictionnaire est correctement chargé.

- 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
```

Référence : SPELL Lire dictionnaire courant

SPELL Lire dictionnaire courant

SPELL Lire dictionnaire courant → Entier long

Paramètres	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	ID du dictionnaire utilisé pour la correction orthographique

La nouvelle commande SPELL Lire dictionnaire courant retourne le numéro d'ID du dictionnaire en cours d'utilisation (Cordial ou Hunspell).

- 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 //196608 par exemple
$countryName:= $Noms_at{Chercher dans tableau($IDs_al;
                                              $curLangCode)}
  // Affichage du message
ALERTE("Dictionnaire courant : "+$countryName) // Espagnol
```

Référence : SPELL FIXER DICTIONNAIRE COURANT

SPELL VERIFIER TEXTE

SPELL VERIFIER TEXTE(leTexte; posErr; longErr; posVérif; tabSuggest)

Paramètres	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	Tab Texte	←	Liste des suggestions

La nouvelle 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.

- 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)
```

Référence : SPELL CORRECTION ORTHOGRAPHIQUE

Définition structure

FIXER CHEMIN DONNEES EXTERNES

FIXER CHEMIN DONNEES EXTERNES(leChamp; chemin)

Paramètres	Type		Description
leChamp	Champ Texte, Blob ou Image	→	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

La nouvelle 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.

A compter de 4D v13, le stockage à l'extérieur du fichier de données est possible pour les champs de type Texte, Blob et Image. Pour une description complète de cette fonctionnalité, reportez-vous au paragraphe "Stockage externe des données", page 24.

Le paramétrage défini par cette commande sera appliqué uniquement lors du stockage sur disque de l'enregistrement courant. Les paramètres de stockage définis dans la structure de l'application ne sont pas modifiés. Si l'enregistrement courant est annulé, la commande ne fait rien.

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).

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. paragraphe "Mode automatique ou personnalisé", page 24).
  Vous pouvez passer un chemin relatif au fichier de données ou un chemin absolu, incluant le nom du fichier de stockage et son extension. L'extension doit correspondre au type réel des données (il n'y a pas de conversion au moment du stockage). Vous devez utiliser la syntaxe système. 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. Si un dossier n'existe pas, 4D le créera automatiquement (une erreur est retournée si la création échoue, par exemple si les droits sont insuffisants).
  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 *(valeur)*	Description
Utiliser définition structure (0)	4D utilisera les paramètres définis dans la structure pour le stockage du champ (cf. paragraphe "Configurer le stockage externe dans l'éditeur de structure", page 27). Si vous passez d'un stockage externe à un stockage interne, le fichier externe n'est pas supprimé.
Utiliser dossier par défaut (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.

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.

- Vous souhaitez stocker le contenu d'un champ image à l'extérieur des données s'il dépasse une certaine taille. Dans le bouton de validation du formulaire, vous pouvez écrire :

```
Si (Taille image ([Photos]InputField) > 1300000)
    // stocker les gros fichiers hors data
  FIXER CHEMIN DONNEES EXTERNES ([Photos]InputField ;
    "C:\\Storage\\LargePicts\\" + Chaine(Numero enregistrement)
    + ".jpg")
Sinon
    // stocker les fichiers de plus petite taille comme défini en structure
  FIXER CHEMIN DONNEES EXTERNES ([Photos]InputField ; Utiliser définition structure)
Fin de si
```

Référence : Lire chemin donnees externes

Lire chemin donnees externes

Lire chemin donnees externes (leChamp) → Texte

Paramètres	Type		Description
leChamp	Champ Texte, Blob ou Image	→	Champ dont vous souhaitez obtenir le lieu de stockage
Résultat	Texte	←	Chemin d'accès complet du fichier de stockage externe

La nouvelle 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.

Cette commande vous permet notamment de recopier le fichier externe.

Note Pour plus d'informations sur le stockage externe en v13, reportez-vous au paragraphe "Stockage externe des données", page 24.

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 la valeur Null, auquel cas le fichier externe n'a pas été créé.

Référence : FIXER CHEMIN DONNEES EXTERNES

RECHARGER DONNEES EXTERNES

RECHARGER DONNEES EXTERNES(leChamp)

Paramètres	Type		Description
leChamp	Champ Texte, Blob ou Image	→	Champ pour lequel définir le lieu de stockage

La nouvelle 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 ou Texte.

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 pour deux raisons :

- mettre à jour le contenu du champ s'il est affiché dans un formulaire.
- éviter que lors de la sauvegarde de l'enregistrement l'image modifiée sur disque ne soit écrasée par l'"ancienne" version qui se trouvait en mémoire.

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).

Documents système

COPIER DOCUMENT

COPIER DOCUMENT (source; dest{; *})

Paramètres	Type		Description
source	Chaîne	→	Chemin d'accès du fichier *ou du dossier* à copier
dest	Chaîne	→	Nom ou chemin d'accès du fichier *ou du dossier* copié
*		→	Remplacer le document existant le cas échéant

La commande existante COPIER DOCUMENT prend désormais en charge la copie de dossiers. Vous pouvez utiliser cette commande pour recopier par programmation l'intégralité d'un dossier ou pour recopier un fichier dans un dossier.

Pour indiquer que vous désignez un dossier, les chaînes passées dans *source* et *dest* 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 *source*. Ce dossier doit exister sur le disque.

Lorsqu'un dossier est défini dans le paramètre *source*, un dossier doit également être désigné dans le paramètre *dest*. Vous devez passer un chemin d'accès complet de dossier (dont les composantes doivent déjà exister sur le disque).

Si un dossier du même nom que celui désigné par le paramètre *source* existe déjà à l'emplacement défini par *dest* 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 fichier à l'emplacement de destination dans ce cas.

A noter que vous pouvez passer un fichier dans le paramètre *source* et un dossier dans le paramètre *dest*, afin de copier un fichier dans un dossier.

- Copie d'un fichier dans un dossier spécifique en conservant le même nom :

  COPIER DOCUMENT("C:\\Projets\\NomDoc";"C:\\Projets\\")

- 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\\"; *)

- Copie d'un dossier dans un autre dossier (les deux dossiers doivent exister sur le disque) :

  COPIER DOCUMENT("C:\\Projets\\";"C:\\Archives\\2011\\")

CREER DOSSIER

CREER DOSSIER(cheminAccès*{; *}*)

Paramètres	Type		Description
cheminAccès	Chaîne	→	Chemin d'accès du nouveau dossier à créer *ou du fichier*
*		→	*Créer la hiérarchie du dossier*

La commande existante CREER DOSSIER permet désormais de recréer une hiérarchie de dossiers lorsqu'elle n'existe pas déjà. Pour cela, il vous suffit de passer le paramètre *.

Vous pouvez également passer un chemin d'accès de document dans *cheminAccès*. Dans ce cas, le nom du document est ignoré mais la hiérarchie de dossiers définie dans *cheminAccès* est créée récursivement.

- Création de la hiérarchie de dossiers "C:\Archives\2011\January\" :

  CREER DOSSIER ("C:\\Archives\\2011\\January\\";*)

- 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
```

LISTE DES DOCUMENTS

LISTE DES DOCUMENTS (cheminAccès; documents*{; options}*)

Paramètres	Type		Description
cheminAccès	Texte	→	Chemin d'accès de volume ou de dossier
documents	Tab Texte	→	Noms des documents
options	*Entier long*	→	*Options de construction de la liste*

La commande existante LISTE DES DOCUMENTS admet un paramètre optionnel supplémentaire, *options*, permettant de définir les informations à faire figurer dans le tableau *documents*.

Pour définir le paramètre *options*, utilisez les nouvelles constantes suivantes, placées dans le thème "Documents système" (vous pouvez passer une combinaison de constantes) :

Constante (valeur)	Description
Chemin récursif (1)	Le tableau documents contient les fichiers et tous les sous-dossiers du dossier spécifié
Chemin absolu (2)	Le tableau documents contient des chemins d'accès absolus
Chemin POSIX (4)	Le tableau documents contient des chemins d'accès au format POSIX
Ignorer invisibles (8)	Les documents invisibles ne sont pas listés

Note - 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 "/"

- Liste de tous les documents dans un dossier (syntaxe précédente) :

```
LISTE DES DOCUMENTS("C:\";tabFichiers)
-> TabFichiers :
     Texte1.txt
     Texte2.txt
```

- 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

- 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

- 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

- 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

Selectionner document

Selectionner document (répertoire ; typesFichiers ; titre ; *options* {; sélectionnés}) → Texte

Une nouvelle constante est disponible pour le paramètre *options* (paramètre permettant de spécifier les fonctions avancées autorisées dans la boîte de dialogue d'ouverture). Cette constante est placée dans le thème "Documents système" :

Saisie nom fichier (32)	Autorise l'utilisateur à saisir un nom de fichier dans la boîte de dialogue. 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)

- Création d'un document personnalisé par l'utilisateur :

```
$doc:= Selectionner document (Dossier système(Dossier documents)+
            "Report.pdf";"pdf";"Nom de l'état :"; Saisie nom fichier)
Si(OK=1)
  BLOB VERS DOCUMENT(Document;$blob) // $blob contient le document à enregistrer
Fin de si
```

Constante renommée

Dans le thème "Documents système", la constante existante Est un répertoire a été renommée **Est un dossier** dans 4D v13. Ce nouveau nom est plus conforme aux interfaces Mac OS et Windows actuelles.

Environnement 4D

Compacter fichier donnees

Compacter fichier donnees (cheminStructure ; cheminDonnées {; dossierArchive{; options{; méthode}}}) → Texte

Le paramètre facultatif *options* admet deux options supplémentaires, accessibles via deux nouvelles constantes du thème "Maintenance fichier de données" :

Constante (valeur)	Commentaire
Mettre à jour enregistrements (65536)	Forcer la réécriture de tous les enregistrements suivant la définition courante des champs dans la structure
Compacter table adresses (131072)	Forcer la réécriture de la table d'adresses des enregistrements (ralentit le compactage). Si vous passez uniquement cette option, 4D active automatiquement l'option 'Mettre à jour enregistrements'.

Pour plus d'informations sur ces options, reportez-vous au paragraphe "Centre de Sécurité et de Maintenance", page 60.

CREER FICHIER DONNEES

CREER FICHIER DONNEES (cheminAccès)

4D Server : La commande CREER FICHIER DONNEES peut désormais 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.

Référence : OUVRIR FICHIER DONNEES

Dossier 4D

Dossier 4D ({dossier}{;}{*}) → Chaîne

Sous Mac OS, l'emplacement retourné par la commande avec la constante Dossier 4D actif en paramètre (ou sans paramètre) a été modifié dans 4D v13.

Le nouvel emplacement retourné est :
{Disque}:Users:*Utilisateur*:Library:Application Support:4D

Note de compatibilité Dans les versions précédentes de 4D sous Mac OS, le dossier 4D actif était situé à l'emplacement [...]Library:Preferences:4D
Si vous avez converti une base depuis une ancienne version et si cette base accède à des contenus personnalisés stockés dans le dossier 4D actif, veillez à les recopier manuellement ou par programmation dans le nouvel emplacement.

FIXER PARAMETRE BASE, Lire parametre base

FIXER PARAMETRE BASE ({table; }sélecteur; valeur)

Lire parametre base ({table; }sélecteur{; valeurAlpha}) → Réel

Les commandes FIXER PARAMETRE BASE, Lire parametre base admettent de nouveaux sélecteurs permettant de paramétrer l'implémentation du moteur de rendu graphique Direct2D dans 4D sous Windows.
Une nouvelle valeur pour le sélecteur Enreg événements debogage et deux nouveaux sélecteurs (Enreg diagnostic et Liste commandes enreg) complètent les outils d'aide à la mise au point des applications.
Enfin, les sélecteurs de gestion du serveur Web deviennent obsolètes.

Gestion de Direct2D

Le moteur de rendu Direct2D est pris en charge à partir de Windows Vista uniquement.

Note Pour plus d'informations sur Direct2D, reportez-vous au paragraphe "Nouveau moteur de rendu Direct2D sous Windows", page 51.

- **Sélecteur = 69** (Direct2D Statut)
  - *Description* : Mode d'activation de l'implémentation de Direct2D sous Windows.
  - *Valeurs* : Une des constantes suivantes du thème "Paramètres de la base" (mode 5 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 Matériel SVG et Editeurs (2) : utilisation de Direct2D en contexte graphique matériel uniquement pour le SVG ainsi que pour les éditeurs de code et de structure. 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) : 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.
    • Direct2D Logiciel SVG et Editeurs (4) : à partir de Windows 7, utilisation de Direct2D en contexte graphique logiciel uniquement pour le SVG ainsi que pour les éditeurs de code et de structure. Sous Vista, pour des raisons de performances le mode GDI/GDIPlus est utilisé.
    • Direct2D Hybride (5) (*Mode par défaut*) : à partir de Windows 7, utilisation de Direct2D en contexte graphique matériel pour le SVG ainsi que pour les éditeurs de code et de structure, et utilisation de Direct2D en contexte graphique logiciel pour le reste de l'application 4D. Sous Vista, pour des raisons de performances le mode GDI/GDIPlus est utilisé.

- **Sélecteur = 74** (Direct2D Lire statut actif)
  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* : 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).

Débogage

- **Sélecteur = 34** (Enreg événements débogage)
  Ce sélecteur admet une nouvelle valeur : 4 = enregistrer en mode détaillé avec temps d'exécution.
  Lorsque ce mode est activé, une information supplémentaire est ajoutée dans le fichier de débogage : les temps d'exécution des opérations en millisecondes. Par exemple :
  4072 p=5 puid=16 (2) cmd: DELAY PROCESS(5;60). *1046 ms*
  5335 p=5 puid=17 end_meth: Method2 *5335 ms*

  La valeur "< ms" est affichée si une commande s'exécute en moins d'une milliseconde.
  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 nouveau sélecteur 80, Liste commandes enreg (cf. ci-dessous).

- **Sélecteur = 79** (Enreg diagnostic)
  - *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.
  - *Valeurs* : 0 ou 1 (0 = ne pas enregistrer, 1 = enregistrer)
- **Sélecteur = 80** (Liste commandes enreg)
  - *Description* : Liste des commandes 4D à enregistrer dans le fichier de débogage (cf. sélecteur 34, Enreg événements débogage). 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.
  - *Valeurs* : 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.

▸ Exemples :

```
FIXER PARAMETRE BASE(Liste commandes enreg;"277;341") // enregistrer
    // uniquement les commandes CHERCHER
    // et CHERCHER DANS SELECTION

FIXER PARAMETRE BASE(Liste commandes enreg;"") // Ne pas enregistrer
    // de commandes 4D
```

Report des sélecteurs Web

Les sélecteurs dédiés à la gestion du serveur Web sont désormais conservés par compatibilité uniquement. À compter de la 4D v13, toutes les options liées au serveur Web doivent être gérées via les

nouvelles commandes WEB FIXER OPTION et WEB LIRE OPTION. Les sélecteurs déplacés sont :

Sélecteur	Valeur	Nouveau sélecteur commandes WEB FIXER OPTION / WEB LIRE OPTION
Adresse IP d'écoute	16	Web Adresse IP d'écoute
Enreg requêtes Web	29	Web Enreg requêtes
Jeu de caractères	17	Web Jeu de caractères
Niveau de compression HTTP	50	Web Niveau de compression HTTP
Numéro de port HTTPS	39	Web Numéro de port HTTPS
Process Web simultanés maxi	18	Web Process Web simultanés maxi
Seuil de compression HTTP	51	Web Seuil de compression HTTP
Taille maximum requêtes Web	27	Web Taille max requêtes

L'action de ces sélecteurs est inchangée. À noter que les nouvelles options définies par la commande WEB FIXER OPTION sont toujours locales au serveur Web et valides durant la session uniquement (elles ne sont pas conservées).

LIRE STATISTIQUES MEMOIRE

LIRE STATISTIQUES MEMOIRE est le nouveau nom de la commande LIRE STATISTIQUES CACHE dans 4D v13. Cette commande a été renommée pour une meilleure adéquation avec le type d'informations qu'elle permet d'obtenir.

Le fonctionnement de la commande est inchangé.

OUVRIR FENETRE PROPRIETES

OUVRIR FENETRE PROPRIETES (sélecteur {; accès} *{; typePropriétés}*)

Paramètres	Type		Description
sélecteur	Chaîne	→	Clé de thème ou de page de la boîte de dialogue des Préférences ou des Propriétés de la base
accès	Booléen	→	Vrai=Verrouiller les autres pages, Faux ou omis=Laisser actives les autres pages
typePropriétés	*Entier long*	→	*0 ou omis = Propriétés structure, 1 = Propriétés utilisateur*

Note Cette commande était nommée OUVRIR PREFERENCES 4D dans les versions précédentes de 4D.

La commande OUVRIR FENETRE PROPRIETES accepte un paramètre optionnel supplémentaire, *typePropriétés*.

Ce paramètre est pris en compte dans les bases configurées en mode "Propriétés utilisateur" uniquement (cf. paragraphe “Externaliser des propriétés utilisateur”, page 62). Dans ce contexte, ce paramètre vous permet d’indiquer si vous souhaitez accéder à la boîte de dialogue des "propriétés structure" ou des "propriétés utilisateur". Vous pouvez passer une des nouvelles constantes suivantes, placées dans le thème "Environnement 4D" :

Constante (valeur)	**Description**
Propriétés structure (0)	Accès aux "propriétés structure" (valeur par défaut si paramètre omis). Dans ce mode, les valeurs de *sélecteur* utilisables sont identiques à celles du mode standard.
Propriétés utilisateur (1)	Accès aux "propriétés utilisateur". Dans ce mode, seules certaines clés sont utilisables dans le paramètre *sélecteur* (cf. ci-dessous)

En mode "Propriétés utilisateur", vous devez passer dans *sélecteur* une des clés suivantes :

/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

OUVRIR FICHIER DONNEES

OUVRIR FICHIER DONNEES (cheminAccès)

4D Server : La commande OUVRIR FICHIER DONNEES peut désormais 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.

Référence : CREER FICHIER DONNEES

Environnement système

Dossier systeme

Dossier systeme {(type)} → Texte

Deux nouveautés concernent la commande existante Dossier systeme :

- Une nouvelle constante, Dossier documents, peut être utilisée dans le paramètre *type*. Elle est disponible dans le thème "Dossier Système" :

Constante (valeur)	**Description**
Dossier documents (17)	Dossier "Documents" de l'utilisateur

- Sous Mac OS, de nouvelles valeurs sont retournées par la commande avec les constantes *type* suivantes :
  - Préférences utilisateurs
  - Préférences utilisateurs_tous

Désormais l'emplacement du dossier ~/Library/Application Support/ est retourné (~/Library/Application Support/MyApp/Preferences pour une application fusionnée avec 4D Engine).
Dans les versions précédentes de 4D, par défaut les emplacements ~/Library/Preferences/ étaient retournés.

ENREGISTRER EVENEMENT

ENREGISTRER EVENEMENT ({typeSortie ;} message {; importance})

La nouvelle constante Vers historique diagnostic peut être utilisée dans le paramètre *typeSortie* :

Constante (valeur)	**Description**
Vers historique diagnostic (5)	Indique à 4D d'inscrire le *message* dans le fichier de diagnostic de 4D, si ce fichier a été activé.

Le fichier d'historique de diagnostic peut être activé à l'aide de la commande FIXER PARAMETRE BASE (cf. FIXER PARAMETRE BASE, Lire parametre base).

Evénements formulaire

Evenement formulaire

Evenement formulaire → Entier long

Deux nouveaux événements formulaire ont été ajoutés dans 4D v13 et les événements **Sur déployer** et **Sur contracter** sont désormais disponibles avec les list box hiérarchiques.

Nouveaux événements

Plusieurs nouvelles constantes d'événement formulaire sont proposées dans 4D v13 :

Constante	Type	Valeur	Commentaire
Sur changement page	Entier long	56	On a changé de page courante dans le formulaire
Sur clic pied	Entier long	57	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 uniquement*) L'utilisateur a demandé à supprimer un élément

- **Sur clic pied** : cet événement est disponible pour les objets de type List box et colonne de list box uniquement (il est appelé dans les méthodes objet de ces objets).
  Lorsque cet événement se produit, 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.
  Pour plus d'informations sur la zone de pied des list box, reportez-vous au paragraphe "Pieds de page", page 32.

- **Sur changement page** : cet événement est disponible au niveau des formulaires (il est appelé dans la méthode formulaire).
  Cet événement 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.

- **Sur action suppression** : cet événement est disponible pour les listes hiérarchiques et les list box. Il 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**.
  Pour les list box, l'événement est disponible uniquement au niveau de l'objet list box et est généré lorsque l'utilisateur tente de supprimer une ligne.
  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).

▸ Exemple de code de gestion d'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
```

Extension des événements existants

Les événements formulaire **Sur déployer** et **Sur contracter** sont désormais utilisables avec les list box hiérarchiques. Ces événements permettent d'optimiser l'utilisation des list box hiérarchiques basées sur des champs dont les valeurs sont chargées à la volée dans des tableaux. Pour plus d'informations, reportez-vous au paragraphe "Nouveaux événements pour les list box hiérarchiques", page 42.

Fenêtres

CHANGER COORDONNEES FENETRE

CHANGER COORDONNEES FENETRE(gauche ; haut ; droite ; bas{; fenêtre}*{; *}*)

Paramètres	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
*	*	→	*Si omis (défaut) = passer la fenêtre au premier plan ;* *Si passé = ne pas changer le plan de la fenêtre*

La commande existante CHANGER COORDONNEES FENETRE admet un nouveau paramètre optionnel *****. Par défaut, lorsque ce paramètre est omis (comme dans les versions précédentes de 4D), l'exécution de la commande fait automatiquement passer au premier plan la fenêtre désignée par le paramètre *fenêtre*, si ce paramètre est utilisé. Désormais, si vous passez le paramètre *****, la commande ne modifie pas le plan initial de la fenêtre (coordonnée "z").

Fonctions statistiques

Dans 4D v13, les capacités des commandes du thème "Fonctions statistiques" ont été étendues : elles peuvent désormais être utilisées avec les tableaux numériques (y compris les tableaux à 2 dimensions).

Ecart type

Ecart type (séries) → Réel

Paramètres	Type		Description
séries	Champ | *Tableau*	→	Données dont vous voulez obtenir l'écart type
Résultat	Réel	←	Ecart type de séries

La commande Ecart type peut désormais être utilisée avec un paramètre *séries* de type tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

- 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) → Réel

Paramètres	Type		Description
séries	Champ | *Tableau*	→	Données dont vous voulez obtenir la valeur la plus élevée
Résultat	Réel	←	Valeur la plus élevée de séries

La commande Max peut désormais être utilisée avec un paramètre *séries* de type tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

- 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)
```

Min

Min (séries) → Réel

Paramètres	Type		Description
séries	Champ | *Tableau*	→	Données dont vous voulez obtenir la valeur la plus basse
Résultat	Réel	←	Valeur la plus basse de séries

La commande Min peut désormais être utilisée avec un paramètre *séries* de type tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

- 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)
```

Moyenne

Moyenne (séries) → Réel

Paramètres	Type		Description
séries	Champ | *Tableau*	→	Données dont vous voulez calculer la moyenne
Résultat	Réel	←	Moyenne de séries

La commande Moyenne peut désormais être utilisée avec un paramètre *séries* de type tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

- 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)
```

Somme

Somme (séries) → Réel

Paramètres	Type		Description
séries	Champ | *Tableau*	→	Données dont vous voulez calculer la somme
Résultat	Réel	←	Somme de séries

La commande Somme peut désormais être utilisée avec un paramètre *séries* de type tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

- 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)
```

Somme des carres

Somme des carres (séries) → Réel

Paramètres	Type		Description
séries	Champ | *Tableau*	→	Données dont vous voulez obtenir la somme des carrés
Résultat	Réel	←	Somme des carrés de séries

La commande Somme des carres peut désormais être utilisée avec un paramètre *séries* de type tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

- 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) → Réel

Paramètres	Type		Description
séries	Champ | *Tableau*	→	Données dont vous voulez obtenir la variance
Résultat	Réel	←	Variance de séries

La commande Variance peut désormais être utilisée avec un paramètre *séries* de type tableau (à une ou deux dimensions). Dans ce cas, le tableau doit être de type Entier, Entier long ou Réel.

- 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

FORM ALLER A PAGE

FORM ALLER A PAGE (numéroPage *{; *}*)I

Paramètres	Type		Description
numéroPage	Entier long	→	Numéro de la page à afficher
*		→	*Changer la page du sous-formulaire courant*

La commande existante FORM ALLER A PAGE accepte désormais un paramètre * optionnel.

Ce 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.

FORM Lire page courante

FORM Lire page courante *{(*)}* → Entier long

Paramètres	Type		Description
*		→	*Retourner le numéro de la page du sous-formulaire courant*
Résultat	Entier long	←	Numéro de la page courante du formulaire actuellement affiché

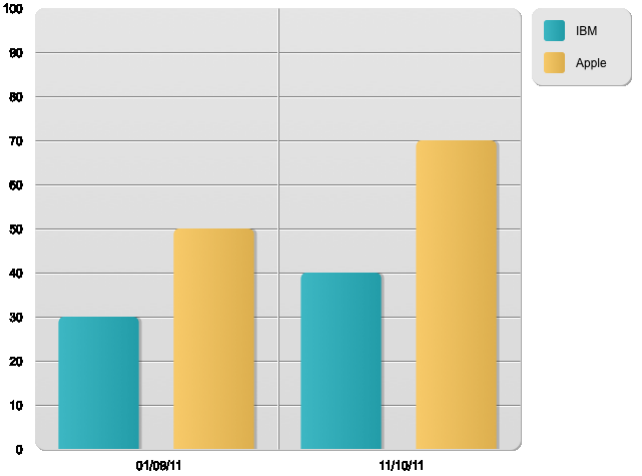
La commande existante FORM Lire page courante accepte désormais un paramètre * optionnel.

Ce paramètre est utile lorsque la commande est appelée dans le contexte d'un sous-formulaire en page. 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.

Graphes

GRAPHE

L'apparence des graphes dessinés via le moteur de rendu SVG de 4D (en cas d'utilisation d'une variable image) a été retravaillée et actualisée. Vous pouvez désormais générer des graphes du type suivant :

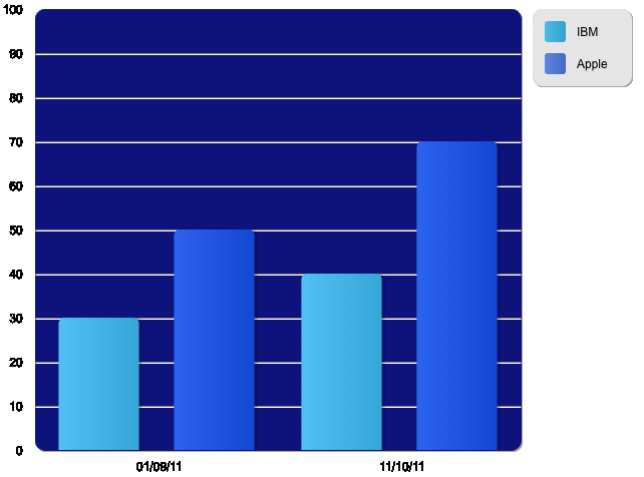


De plus, des IDs spécifiques sont désormais automatiquement attribués aux éléments présents dans le graphe SVG :

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_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

A l'aide des commandes SVG intégrées de 4D, il est alors facile de transformer le graphe ci-dessus afin d'obtenir cette version :



Images

FIXER NOM FICHIER IMAGE

FIXER NOM FICHIER IMAGE(image; nomFichier)

Paramètres	Type		Description
image	Champ ou Variable image	→	Image dont vous souhaitez fixer le nom par défaut
nomFichier	Texte	→	Nom par défaut de l'image

La nouvelle commande FIXER NOM FICHIER IMAGE vous permet d'ajouter ou de modifier le nom de fichier par défaut de l'image passée en paramètre.

Ce nom par défaut est utilisé en cas d'exportation de l'image dans un fichier disque. Il peut avoir été défini automatiquement à partir du nom d'origine du fichier image importé dans le champ ou la variable image. Pour plus d'informations, reportez-vous au paragraphe "Noms par défaut des fichiers image", page 22.

Référence : Lire nom fichier image

Images egales

Images egales (image1; image2; masque) → Booleen

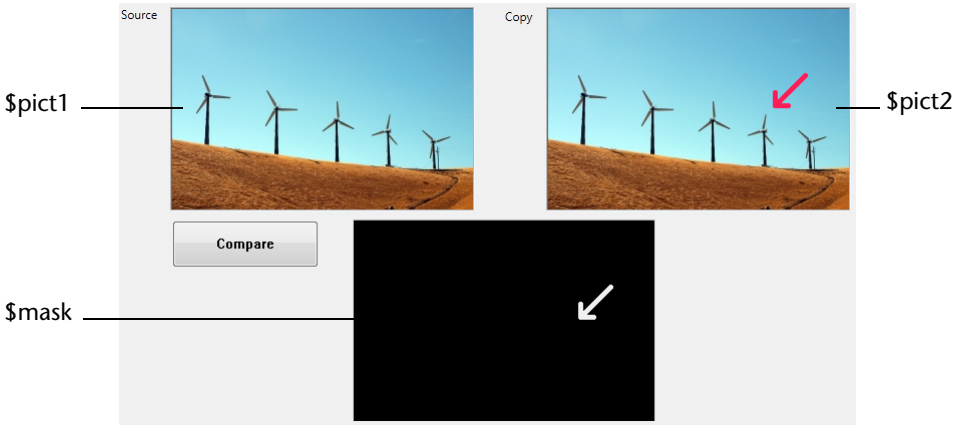
Paramètres	Type		Description
image1	Champ ou Variable image	→	Image source originale
image2	Champ ou Variable image	→	Image à comparer
masque	Champ ou Variable image	←	Masque résultant
Résultat	Booléen	←	Vrai si les deux images sont identiques, sinon Faux

La nouvelle 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.

- 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 MOTS CLES IMAGE

LIRE MOTS CLES IMAGE(image; tabMotsclés{; *})

Paramètres	Type		Description
image	Champ ou Variable image	→	Image dont vous souhaitez lire les mots-clés associés
tabMotsclés	Tab texte	←	Tableau contenant les mots-clés extraits
*	*	→	Si passé = utiliser les valeurs distinctes

La nouvelle 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 désormais indexables dans 4D v13 (cf. paragraphe "Indexation des mots-clés d'images", page 19).

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".

Lire nom fichier image

Lire nom fichier image (image) → Texte

Paramètres	Type		Description
image	Champ ou Variable image	→	Image dont vous souhaitez obtenir le nom par défaut
Résultat	Texte	←	Nom par défaut du fichier image

La nouvelle commande Lire nom fichier image retourne le nom par défaut courant de l'image passée en paramètre.

Le nom par défaut est utilisé en cas d'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 nouvelle commande FIXER NOM FICHIER IMAGE. Pour plus d'informations, reportez-vous au paragraphe "Noms par défaut des fichiers image", page 22.

Si l'*image* n'a pas de nom par défaut, la commande retourne une chaîne vide.

Référence : FIXER NOM FICHIER IMAGE

Impressions

Deux nouvelles commandes permettent un contrôle étendu de l'impression écran.

A noter qu'à compter de la version 13, 4D utilise la technologie XPS (*XML Paper Specification*) pour la génération de documents écran imprimables sous Windows (sous Mac OS, 4D utilise le PDF intégré). Pour plus d'informations sur la prise en charge du XPS dans 4D v13, reportez-vous au paragraphe "Aperçu avant impression XPS sous Windows", page 52.

Lire apercu impression

Lire apercu impression → Booleen

Paramètres	Type		Description
Cette commande ne requiert pas de paramètres			
Résultat	Booleen	←	Vrai = Impression à l'écran, Faux = Pas d'impression écran

La nouvelle commande Lire apercu impression retourne Vrai si l'instruction FIXER APERCU IMPRESSION(Vrai) a été appelée 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 nouvelle commande Est en apercu impression.

Référence : Est en apercu impression

Est en apercu impression

Est en apercu impression → Booleen

Paramètres	Type		Description
Cette commande ne requiert pas de paramètres			
Résultat	Booleen	←	Vrai = Impression à l'écran, Faux = Pas d'impression écran

La nouvelle 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.

- 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
    FORMULAIRE SORTIE([Factures] ;"versEcran")
  Sinon
```

```
    FORMULAIRE SORTIE([Factures] ;"versImprimante"
  Fin de si
  IMPRIMER SELECTION ([Factures])
  Si(Est en apercu impression = Faux)
    [Factures]Imprimée := Vrai
    STOCKER ENREGISTREMENT ([Factures])
    vImp:= vImp+1   // Mise à jour du compteur d'impressions
  Fin de si
Fin de si
```

Référence : Lire apercu impression

List Box

De nombreuses modifications ont été apportées au thème de commandes des List box. De nouvelles commandes ont été ajoutées et des commandes existantes ont été modifiées.

Note Le nouvel événement formulaire **Sur action suppression** est également disponible pour les list box (cf. paragraphe "Evenement formulaire", page 136).

Nouvelles commandes

LISTBOX FIXER CALCUL PIED

LISTBOX FIXER CALCUL PIED ({*; }objet; calcul)

Paramètres	Type		Description
*		→	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

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 nouveau thème "List box pied calcul", pour définir le calcul à effectuer :

calcul **(valeur)**	**Commentaire**
Listbox pied personnalisé (1)	Aucun calcul n'est effectué par 4D. La variable du pied doit être calculée par programmation.
Listbox pied min (2)	Utilisable avec les colonnes de type numérique, date, heure, booléen
Listbox pied max (3)	Utilisable avec les colonnes de type numérique, date, heure, booléen
Listbox pied somme (4)	Utilisable avec les colonnes de type numérique, heure, booléen
Listbox pied nombre (5)	Utilisable avec les colonnes de type numérique, texte, date, heure, booléen, image
Listbox pied moyenne (6)	Utilisable avec les colonnes de type numérique, heure
Listbox pied écart type (7)	Utilisable avec les colonnes de type numérique, heure (listbox de type tableau uniquement)
Listbox pied variance (8)	Utilisable avec les colonnes de type numérique, heure (listbox de type tableau uniquement)
Listbox pied somme des carrés (9)	Utilisable avec les colonnes de type numérique, heure (listbox de type tableau uniquement)

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 Listbox 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.

Référence : LISTBOX Lire calcul pied

LISTBOX FIXER COLONNES STATIQUES

LISTBOX FIXER COLONNES STATIQUES({*; }objet; nbColonnes)

Paramètres	Type		Description
*		→	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

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 ne peuvent pas être déplacées dans une list box. Cette fonctionnalité était déjà présente dans les versions précédentes de 4D, mais uniquement via la Liste des propriétés (elle était nommée "colonnes fixes").

Note Les colonnes statiques et les colonnes verrouillées sont deux fonctionnalités indépendantes. Pour plus d'informations, reportez-vous au paragraphe "Colonnes verrouillées", page 40.

Référence : LISTBOX Lire colonnes statiques

LISTBOX FIXER COLONNES VERROUILLEES

LISTBOX FIXER COLONNES VERROUILLEES({*; }objet; nbColonnes)

Paramètres	Type		Description
*		→	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

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 paragraphe "Colonnes verrouillées", page 40.

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 toute valeur entre 1 et le nombre total de colonnes de la list box dans *nbColonnes*. Pour supprimer le verrouillage de colonnes, passez 0 dans *nbColonnes*.

Référence : LISTBOX Lire colonnes verrouillees

LISTBOX FIXER FORMULE COLONNE

LISTBOX FIXER FORMULE COLONNE ({*; }objet; formule; typeDonnées)

Paramètres	Type		Description
*		→	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

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.

Référence : LISTBOX Lire formule colonne

LISTBOX FIXER HAUTEUR ENTETES

LISTBOX FIXER HAUTEUR ENTETES ({*; }objet; hauteur{; unité})

Paramètres	Type		Description
*		→	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

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" :

***unité* (valeur)**	**Commentaire**
Listbox pixels (0)	La hauteur est un nombre de pixels (défaut)
Listbox lignes (1)	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police

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.

Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au paragraphe "Hauteur des lignes, en-têtes et pieds", page 36.

Référence : LISTBOX Lire hauteur entetes

LISTBOX FIXER HAUTEUR PIEDS

LISTBOX FIXER HAUTEUR PIEDS ({*; }objet; hauteur{; unité})

Paramètres	Type		Description
*		→	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

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" :

***unité* (valeur)**	**Commentaire**
Listbox pixels (0)	La hauteur est un nombre de pixels (défaut)
Listbox lignes (1)	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police

Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au paragraphe "Hauteur des lignes, en-têtes et pieds", page 36.

Référence : LISTBOX Lire hauteur pieds

LISTBOX Lire calcul pied

LISTBOX Lire calcul pied({*; }objet) → Entier long

Paramètres	Type		Description
*		→	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

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 "Listbox Pied calcul" (cf. paragraphe “LISTBOX FIXER CALCUL PIED”, page 149).

Référence : LISTBOX FIXER CALCUL PIED

LISTBOX Lire colonnes statiques

LISTBOX Lire colonnes statiques ({*; }objet) → Entier long

Paramètres	Type		Description
*		→	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

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 paragraphe “Colonnes verrouillées”, page 40.

Référence : LISTBOX FIXER COLONNES STATIQUES

LISTBOX Lire colonnes verrouillees

LISTBOX Lire colonnes verrouillees ({*; }objet) → Entier long

Paramètres	Type		Description
*		→	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

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 paragraphe "Colonnes verrouillées", page 40.

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.

Référence : LISTBOX FIXER COLONNES VERROUILLEES

LISTBOX LIRE COULEUR GRILLE

LISTBOX LIRE COULEUR GRILLE ({* ;} objet; couleurH; couleurV)

Paramètres	Type		Description
*		→	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

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 formule colonne

LISTBOX Lire formule colonne ({*; }objet) → Chaîne

Paramètres	Type		Description
*		→	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

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.

Référence : LISTBOX FIXER FORMULE COLONNE

LISTBOX LIRE GRILLE

LISTBOX LIRE GRILLE ({* ;} objet; horizontal; vertical)

Paramètres	Type		Description
*		→	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	Var booléenne	←	Vrai = affichée, Faux = cachée
vertical	Var booléenne	←	Vrai = affichée, Faux = cachée

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}) → Entier long

Paramètres	Type		Description
*		→	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

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" :

unité **(valeur)**	**Commentaire**
Listbox pixels (0)	La hauteur est un nombre de pixels (défaut)
Listbox lignes (1)	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police

Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au paragraphe "Hauteur des lignes, en-têtes et pieds", page 36.

Référence : LISTBOX FIXER HAUTEUR ENTETES

LISTBOX Lire hauteur pieds

LISTBOX Lire hauteur pieds ({*; }objet{; unité}) → Entier long

Paramètres	Type		Description
*		→	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

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" :

***unité* (valeur)**	**Commentaire**
Listbox pixels (0)	La hauteur est un nombre de pixels (défaut)
Listbox lignes (1)	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police

Pour plus d'informations sur le calcul des hauteurs de lignes, reportez-vous au paragraphe "Hauteur des lignes, en-têtes et pieds", page 36.

Référence : LISTBOX FIXER HAUTEUR PIEDS

Commandes modifiées

LISTBOX FIXER HAUTEUR LIGNES

LISTBOX FIXER HAUTEUR LIGNES ({*; }objet; hauteur*{; unité}*)

Paramètres	Type		Description
*		→	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*

La commande LISTBOX FIXER HAUTEUR LIGNES admet un paramètre optionnel supplémentaire, *unité*. Ce paramètre permet de définir l'unité dans laquelle vous voulez exprimer la valeur de hauteur de ligne de la listbox. Vous pouvez passer dans le paramètre *unité* l'une des constantes suivantes, placées dans le thème "List Box" :

***unité* (valeur)**	**Commentaire**
Listbox pixels (0)	La hauteur est un nombre de pixels (défaut)
Listbox lignes (1)	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police

Si vous omettez ce paramètre, la hauteur est exprimée en pixels comme dans les versions précédentes de 4D.

Référence : LISTBOX Lire hauteur lignes

LISTBOX FIXER TABLE SOURCE

LISTBOX FIXER TABLE SOURCE ({* ;} objet; numTab | tempo *{; nomSurlignage}*})

Paramètres	Type		Description
*		→	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)
numTab | 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*

La commande LISTBOX FIXER TABLE SOURCE vous permet d'associer un ensemble de surlignage à la list box via le nouveau paramètre *nomSurlignage*. L'ensemble de surlignage est utilisé pour gérer le surlignage des enregistrements par l'utilisateur dans la list box.

Référence : LISTBOX LIRE TABLE SOURCE

LISTBOX INSERER COLONNE

LISTBOX INSERER COLONNE ({*; }objet; positionCol; nomCol; variableCol; nomEnTête; variableEntête*{; nomPied; variablePied}*)

Paramètres	Type		Description
*		→	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	Tab, champ, var	→	Nom de la variable tableau de la colonne ou champ ou variable
nomEnTête	Chaîne	→	Nom d'objet de l'en-tête de la colonne
variableEn-tête	Var Entier	→	Variable d'en-tête de la colonne
nomPied	*Chaîne*	→	*Nom d'objet du pied de la colonne*
variablePied	*Var Entier*	→	*Variable du pied de la colonne*

La commande LISTBOX INSERER COLONNE admet deux paramètres supplémentaires, *nomPied* et *variablePied*, permettant de définir le nom d'objet et la variable du pied de la colonne insérée.

LISTBOX INSERER COLONNE FORMULE

LISTBOX INSERER COLONNE FORMULE({*; }objet; positionCol; nomCol; formule; typeDonnées; nomEntête; variableEntête*{; nomPied; variablePied}*)

Paramètres	Type		Description
*		→	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	Var Entier	→	Variable d'en-tête de la colonne
nomPied	*Chaîne*	→	*Nom d'objet du pied de la colonne*
variablePied	*Var Entier*	→	*Variable du pied de la colonne*

La commande LISTBOX INSERER COLONNE FORMULE admet deux paramètres supplémentaires, *nomPied* et *variablePied*, permettant de définir le nom d'objet et la variable du pied de la colonne insérée.

LISTBOX INSERER LIGNES

LISTBOX INSERER LIGNES ({*; }objet; positionLigne*{; nbLignes}*)

Paramètres	Type		Description
*		→	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*

Note de compatibilité Cette commande était nommée LISTBOX INSERER LIGNE dans les versions précédentes de 4D.

La commande LISTBOX INSERER LIGNES permet désormais d'insérer plusieurs lignes en un seul appel. Pour cela, il vous suffit d'indiquer le nombre de lignes à insérer à l'aide du nouveau paramètre *nbLignes*.

Par défaut, si le paramètre *nbLignes* est omis, une seule ligne est insérée.

Référence : LISTBOX SUPPRIMER LIGNES

LISTBOX Lire hauteur lignes

LISTBOX Lire hauteur lignes ({*; }objet*{; unité}*) → Entier long

Paramètres	Type		Description
*		→	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

La commande LISTBOX Lire hauteur lignes admet un paramètre optionnel supplémentaire, *unité*. Ce paramètre permet de définir l'unité dans laquelle vous voulez obtenir la valeur de hauteur de ligne de la listbox. Vous pouvez passer dans le paramètre *unité* l'une des constantes suivantes, placées dans le thème "List Box" :

***unité* (valeur)**	**Commentaire**
Listbox pixels (0)	La hauteur est un nombre de pixels (défaut)
Listbox lignes (1)	La hauteur désigne un nombre de lignes. 4D calcule la hauteur d'une ligne en fonction de la police

Si vous omettez ce paramètre, la hauteur est retournée en pixels comme dans les versions précédentes de 4D.

Référence : LISTBOX FIXER FORMULE COLONNE

LISTBOX Lire information

LISTBOX Lire information ({*; }objet; info) → Entier long

Deux nouvelles constantes sont disponibles dans le paramètre *info*, afin de pouvoir obtenir des informations sur l'affichage et la hauteur des pieds de colonnes. Ces nouvelles constantes sont placées dans le thème "List Box" :

***info* (valeur)**	**Commentaire**
Listbox affichage pied (8)	0=masqué, 1=affiché
Listbox hauteur pied (9)	Hauteur en pixels

A noter que l'instruction LISTBOX Lire information(vLB;Listbox 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.

LISTBOX LIRE POSITION CELLULE

LISTBOX LIRE POSITION CELLULE({*; }objet; colonne; ligne{; varCol})

Cette commande peut désormais être appelée dans le contexte d'une list box hiérarchique générant l'un des événements formulaire suivants :

- **Sur déployer**
- **Sur contracter**

Pour plus d'informations sur ce point, reportez-vous au paragraphe "Nouveaux événements pour les list box hiérarchiques", page 42.

LISTBOX LIRE TABLE SOURCE

LISTBOX LIRE TABLE SOURCE ({* ;} objet; numTab{; nomSélection*{; nomSurlignage}*})

Paramètres	Type		Description
*		→	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)
numTab	Entier long	←	Numéro de la table de la sélection
nomSélection	Chaîne	←	Nom de la sélection temporaire ou "" pour la sélection courante
nomSurlignage	*Chaîne*	←	*Nom de l'ensemble de surlignage*

La commande LISTBOX LIRE TABLE SOURCE retourne désormais dans le nouveau paramètre *nomSurlignage* le nom de l'ensemble surlignage associé à la listbox. Si aucun ensemble surlignage n'a été défini, la commande retourne une chaîne vide.

Référence : LISTBOX FIXER TABLE SOURCE

LISTBOX LIRE TABLEAUX

LISTBOX LIRE TABLEAUX ({* ;} objet; tabNomsCols; tabNomsEntêtes; tabVarCols; tabVarEntêtes; tabColsVisibles; tabStyles*{; tabNomsPieds; tabVarPieds}*)

Paramètres	Type		Description
*		→	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	Tab chaîne	←	Noms d'objet des colonnes
tabNomsEntêtes	Tab chaîne	←	Noms d'objet des en-têtes
tabVarCols	Tab pointeur	←	Pointeurs vers les variables des colonnes ou Pointeurs vers les champs des colonnes ou Nil
tabVarEntêtes	Tab pointeur	←	Pointeurs vers les variables des en-têtes
tabColsVisibles	Tab booléen	←	Visibilité de chaque colonne
tabStyles	Tab pointeur	←	Pointeurs vers les tableaux ou les variables de styles de couleurs et de visibilité des lignes ou Nil
tabNomsPieds	*Tab chaîne*	←	*Noms d'objet des pieds de colonnes*
tabVarPieds	*Tab pointeur*	←	*Pointeurs vers les variables des pieds de colonnes*

La commande LISTBOX LIRE TABLEAUX retourne désormais les noms d'objet et les variables associées aux zones de pied de la listbox :

- Le tableau *tabNomsPieds* contient la liste des noms d'objet de chaque pied de colonne de la list box.
- Le tableau *tabVarPieds* contient des pointeurs vers les variables associées à chaque pied de colonne de la list box.

LISTBOX SUPPRIMER LIGNES

LISTBOX SUPPRIMER LIGNES ({*; }objet; positionLigne*{; nbLignes}*)

Paramètres	Type		Description
*		→	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 supprimer
nbLignes	*Entier long*	→	*Nombre de lignes à supprimer*

Note de compatibilité Cette commande était nommée LISTBOX SUPPRIMER LIGNE dans les versions précédentes de 4D.

La commande LISTBOX SUPPRIMER LIGNES permet désormais de supprimer plusieurs lignes en un seul appel. Pour cela, il vous suffit d'indiquer le nombre de lignes à supprimer à l'aide du nouveau paramètre *nbLignes*.

Par défaut, si le paramètre *nbLignes* est omis, une seule ligne est supprimée.

Référence : LISTBOX INSERER LIGNES

Commandes renommées

Pour plus de clarté et de cohérence, les commandes existantes suivantes ont été renommées :

Nom versions précédentes	*Nouveau nom v13*
LISTBOX MONTRER GRILLE	**LISTBOX FIXER GRILLE**
LISTBOX INSERER LIGNE	LISTBOX INSERER LIGNES
LISTBOX SUPPRIMER LIGNE	LISTBOX SUPPRIMER LIGNES

Constantes renommées

Par soucis d'harmonisation, les constantes existantes suivantes du thème "List box" ont été renommées :

Nom versions précédentes	*Nouveau nom v13*
Affichage barre déf hor listbox	**Listbox affichage barre déf hor**
Affichage barre déf ver listbox	**Listbox affichage barre déf ver**
Affichage entête listbox	**Listbox affichage entête**
Ajouter à sélection listbox	**Listbox ajouter à sélection**
Hauteur barre déf hor listbox	**Listbox hauteur barre déf hor**
Hauteur entête listbox	**Listbox hauteur entête**
Largeur barre déf ver listbox	**Listbox largeur barre déf ver**
Position barre déf hor listbox	**Listbox position barre déf hor**
Position barre déf ver listbox	**Listbox position barre déf ver**
Remplacer sélection listbox	**Listbox remplacer sélection**
Supprimer de sélection listbox	**Listbox supprimer de sélection**

Listes hiérarchiques

LIRE PARAMETRE ELEMENT TABLEAUX

LIRE PARAMETRE ELEMENT TABLEAUX ({*; }liste; refElément | *; tabSélecteurs{; tabValeurs})

Paramètres	Type		Description
*		→	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, *	→	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

La nouvelle 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 *réfElément* 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").

- 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")
```

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; tValeurs-
Params)
  // tValeursParams{1} contient "Phil"
  // tValeursParams{2} contient "15/02/1978"
  // tValeursParams{3} contient "1"
  // tValeursParams{4} contient "32"
  // tValeursParams{5} contient "Nantes"
```

Méthodes base

Sur événement système

La nouvelle 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 nouveau thème "Evénements de la base" :

Constante (valeur)	Commentaire
Sur passage arrière plan (1)	L'application 4D passe à l'arrière plan
Sur passage premier plan (2)	L'application 4D passe au premier plan

Note 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
```

Sur fermeture session Web

La méthode base Sur fermeture session 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.

Un exemple d'utilisation de la méthode base Sur fermeture session Web est fourni dans le paragraphe "Gestion des sessions Web", page 67.

Outils

Generer digest

Generer digest (param ; algorithme) → Texte

Paramètres	Type		Description
param	BLOB | Text	→	Blob ou texte pour lequel obtenir une clé digest
algorithme	Entier long	→	Algorithme utilisé pour retourner la clé : 0 = Digest MD5, 1 = Digest SHA1
Résultat	Text	←	Valeur de la clé digest

La nouvelle commande Generer digest retourne la clé digest d'un BLOB ou d'un texte après application d'un algorithme de cryptage.

Dans 4D v13, deux algorithmes sont disponibles : **MD5** (*Message Digest 5*) et **SHA-1** (*Secure Hash 1*). 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.

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, quel que soit le mode de fonctionnement de la base de données (Unicode ou compatibilité ASCII).

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 nouveau thème "Type digest" :

Constante (valeur)	Commentaire
Digest MD5 (0)	Utiliser l'algorithme MD5
Digest SHA1 (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.

Note 4D Pack La commande 4D Pack existante AP Get file MD5 digest est désormais obsolète et ne doit plus être utilisée.

▸ 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
```

- 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"
```

TRAITER BALISES 4D

Cette commande était auparavant nommée TRAITER BALISES HTML et était placée dans le thème "Serveur Web". Son fonctionnement est inchangé.

Propriétés des objets

OBJET FIXER ALIGNEMENT VERTICAL

OBJET FIXER ALIGNEMENT VERTICAL ({*; }objet; alignement)

Paramètres	Type		Description
*		→	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

La nouvelle 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 "Propriétés des objets" :

Constante	Type	Valeur
Aligné par défaut	Entier long	1
Aligné en haut	Entier long	2
Aligné au centre	Entier long	3
Aligné en bas	Entier long	4

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.

Référence : OBJET Lire alignement vertical

OBJET FIXER CONFIGURATION CLAVIER

OBJET FIXER CONFIGURATION CLAVIER({*; }objet; codeLangue)

Paramètres	Type		Description
*		→	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

La nouvelle 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.

Note Cette propriété force la prise en compte d'un script de langue lors de la saisie de données. Elle n'est utilisable qu'en mode compatibilité ASCII. En mode Unicode, elle est ignorée.

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 *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 dans le manuel *Langage* de 4D.

Référence : OBJET Lire configuration clavier

OBJET FIXER CORRECTION ORTHOGRAPHIQUE

OBJET FIXER CORRECTION ORTHOGRAPHIQUE({*; }objet; correctionAuto)

Paramètres	Type		Description
*		→	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

La nouvelle 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.

Référence : OBJET Lire correction orthographique

OBJET FIXER EQUIVALENT CLAVIER

OBJET FIXER EQUIVALENT CLAVIER({*; }objet; touche{; modifiers})

Paramètres	Type		Description
*		→	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

La nouvelle 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 nouveau thème "Touches équivalents clavier" (ou sa valeur) :

Constante	Type	Valeur
Raccourci avec F1	Chaîne	[F1]
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 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 Retour Chariot	Chaîne	[return]
Raccourci avec Entrée	Chaîne	[enter]
Raccourci avec Effacement arrière	Chaîne	[backspace]
Raccourci avec Tabulation	Chaîne	[tab]
Raccourci avec Echappement	Chaîne	[esc]
Raccourci avec Suppression	Chaîne	[del]
Raccourci avec Début	Chaîne	[home]
Raccourci avec Fin	Chaîne	[end]
Raccourci avec Aide	Chaîne	[help]
Raccourci avec Page préc	Chaîne	[page up]
Raccourci avec Page suiv	Chaîne	[page down]
Raccourci avec Flèche gauche	Chaîne	[left arrow]
Raccourci avec Flèche droite	Chaîne	[right arrow]
Raccourci avec Flèche haut	Chaîne	[up arrow]
Raccourci avec Flèche bas	Chaîne	[down arrow]

Passez dans le paramètre *modifiers* une ou plusieurs touche(s) de modification à associer à la touche de raccourci.

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.

Pour définir le paramètre *modifiers*, passez une ou plusieurs des constante(s) de type "Masque" suivantes du thème "Evénements (Modifiers)" :

Constante (valeur)	Commentaire
Masque touche majuscule (512)	Windows et Mac OS
Masque touche commande (256)	Windows = touche Ctrl, Mac OS = touche Commande
Masque touche option (2048)	Windows = touche Alt, Mac OS = touche Option
Masque touche contrôle (4096)	Mac OS uniquement

- 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
```

Référence : OBJET LIRE EQUIVALENT CLAVIER

OBJET FIXER MESSAGE AIDE

OBJET FIXER MESSAGE AIDE({*; }objet; messageAide)

Paramètres	Type		Description
*		→	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

La nouvelle 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.

Référence : OBJET Lire message aide

OBJET FIXER OPTIONS GLISSER DEPOSER

OBJET FIXER OPTIONS GLISSER DEPOSER({*; }objet; glissable; glissableAuto; déposable; déposableAuto)

Paramètres	Type		Description
*		→	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	Entier long	→	0 = Faux, 1 = Vrai
glissableAuto	Entier long	→	0 = Faux, 1 = Vrai
déposable	Entier long	→	0 = Faux, 1 = Vrai
déposableAuto	Entier long	→	0 = Faux, 1 = Vrai

La nouvelle 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 une valeur indiquant si l'option correspondante est active (1) ou inactive (0). Vous pouvez utiliser les constantes suivantes du thème "Propriétés des objets".

- *glissable* :

Constante (valeur)	Commentaire
Glissable Vrai (1)	Objet glissable en mode programmé
Glissable Faux (0)	Objet non glissable en mode programmé

- *glissableAuto* (utilisable uniquement avec les champs et variables texte, combo box et list box) :

Constante (valeur)	Commentaire
Glissable auto Vrai (1)	Objet glissable en mode auto
Glissable auto Faux (0)	Objet non glissable en mode auto

- *déposable* :

Constante (valeur)	Commentaire
Déposable Vrai (1)	Objet acceptant le déposer en mode programmé
Déposable Faux (0)	Objet n'acceptant pas le déposer en mode programmé

- *déposableAuto* (utilisable uniquement avec les champs et variables image, texte, combo box et list box) :

Constante (valeur)	Commentaire
Déposable auto Vrai (1)	Objet acceptant le déposer en mode auto
Déposable auto Faux (0)	Objet n'acceptant pas le déposer en mode auto

▸ Définition d'une zone de texte en glisser-déposer auto :

OBJET FIXER OPTIONS GLISSER DEPOSER(*;"Comments";Glissable Faux; Glissable auto Vrai; Déposable Faux; Déposable auto Vrai)

Référence : OBJET LIRE OPTIONS GLISSER DEPOSER

OBJET FIXER RECTANGLE FOCUS INVISIBLE

OBJET FIXER RECTANGLE FOCUS INVISIBLE({*; }objet; invisible)

Paramètres	Type		Description
*		→	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

La nouvelle 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.

Référence : OBJET Lire rectangle focus invisible

OBJET FIXER REDIMENSIONNEMENT

OBJET FIXER REDIMENSIONNEMENT({*; }objet; horizontal; vertical)

Paramètres	Type		Description
*		→	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

La nouvelle 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 "Propriétés des objets" :

Constante (valeur)	Commentaire
Redim horizontal agrandir (1)	Si la fenêtre s'agrandit de 50% en largeur, l'objet s'élargit de 50% vers la droite
Redim horizontal aucun (0)	Si la fenêtre s'agrandit en largeur, ni la largeur ni la position de l'objet ne varient
Redim horizontal déplacer (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 "Propriétés des objets" :

Constante (valeur)	Commentaire
Redim vertical agrandir (1)	Si la fenêtre s'agrandit de 50% en hauteur, l'objet s'allonge de 50% vers le bas
Redim vertical aucun (0)	Si la fenêtre s'agrandit en hauteur, ni la hauteur ni la position de l'objet ne varient
Redim vertical déplacer (2)	Si la fenêtre s'agrandit de 100 pixels en hauteur, l'objet se déplace de 100 pixels vers le bas

Référence : OBJET LIRE REDIMENSIONNEMENT

OBJET FIXER SOUS FORMULAIRE

OBJET FIXER SOUS FORMULAIRE({*; }objet{; laTable}; sousFormDetail{; sousFormListe})

Paramètres	Type		Description
*		→	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)

La nouvelle 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.

Référence : OBJET LIRE SOUS FORMULAIRE

OBJET Lire alignement vertical

OBJET Lire alignement vertical ({*; }objet) → Entier long

Paramètres	Type		Description
*		→	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

La nouvelle 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, placées dans le thème "Propriétés des objets" :

Constante	Type	Valeur
Aligné en haut	Entier long	2
Aligné au centre	Entier long	3
Aligné en bas	Entier long	4

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.

Référence : OBJET FIXER ALIGNEMENT VERTICAL

OBJET Lire configuration clavier

OBJET Lire configuration clavier({*; }objet) → Chaîne

Paramètres	Type		Description
*		→	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

La nouvelle 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.

Note Cette propriété force la prise en compte d'un script de langue lors de la saisie de données. Elle n'est utilisable qu'en mode compatibilité ASCII. En mode Unicode, elle est ignorée.

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 dans le manuel *Langage* de 4D.

Référence : OBJET FIXER CONFIGURATION CLAVIER

OBJET Lire correction orthographique

OBJET Lire correction orthographique({*; }objet) → Booléen

Paramètres	Type		Description
*		→	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

La nouvelle 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.

Référence : OBJET FIXER CORRECTION ORTHOGRAPHIQUE

OBJET LIRE EQUIVALENT CLAVIER

OBJET LIRE EQUIVALENT CLAVIER({*; }objet; touche{; modifiers})

Paramètres	Type		Description
*		→	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

La nouvelle 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 nouveau thème "Touches équivalent 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 (valeur)	Commentaire
Masque touche majuscule (512)	Windows et Mac OS
Masque touche commande (256)	Windows = touche Ctrl, Mac OS = touche Commande
Masque touche option (2048)	Windows = touche Alt, Mac OS = touche Option
Masque touche contrôle (4096)	Mac OS uniquement

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*.

Référence : OBJET FIXER EQUIVALENT CLAVIER

OBJET Lire message aide

OBJET Lire message aide ({*; }objet) → Texte

Paramètres	Type		Description
*		→	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

La nouvelle 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.

- 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
```

Référence : OBJET FIXER MESSAGE AIDE

OBJET LIRE OPTIONS GLISSER DEPOSER

OBJET LIRE OPTIONS GLISSER DEPOSER({*; }objet; glissable; glissableAuto; déposable; déposableAuto)

Paramètres	Type		Description
*		→	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	Entier long	←	0 = Faux, 1 = Vrai
glissableAuto	Entier long	←	0 = Faux, 1 = Vrai
déposable	Entier long	←	0 = Faux, 1 = Vrai
déposableAuto	Entier long	←	0 = Faux, 1 = Vrai

La nouvelle 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 1 ou 0 suivant que l'option correspondante est active (1) ou inactive (0).

Vous pouvez également comparer les valeurs reçues aux constantes suivantes du thème "Propriétés des objets" :

Constante (valeur)	Commentaire
Glissable Vrai (1)	Objet glissable en mode programmé
Glissable Faux (0)	Objet non glissable en mode programmé
Glissable auto Vrai (1)	Objet glissable en mode auto
Glissable auto Faux (0)	Objet non glissable en mode auto
Déposable Vrai (1)	Objet acceptant le déposer en mode programmé
Déposable Faux (0)	Objet n'acceptant pas le déposer en mode programmé

Déposable auto Vrai (1)	Objet acceptant le déposer en mode auto
Déposable auto Faux (0)	Objet n'acceptant pas le déposer en mode auto

Référence : OBJET FIXER OPTIONS GLISSER DEPOSER

OBJET Lire rectangle focus invisible

OBJET Lire rectangle focus invisible({*; }objet) → Booléen

Paramètres	Type		Description
*		→	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

La nouvelle 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.

Référence : OBJET FIXER RECTANGLE FOCUS INVISIBLE

OBJET LIRE REDIMENSIONNEMENT

OBJET LIRE REDIMENSIONNEMENT ({*; }objet; horizontal; vertical)

Paramètres	Type		Description
*		→	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

La nouvelle 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 "Propriétés des objets" :

Constante (valeur)	Commentaire
Redim horizontal agrandir (1)	Si la fenêtre s'agrandit de 50% en largeur, l'objet s'élargit de 50% vers la droite
Redim horizontal aucun (0)	Si la fenêtre s'agrandit en largeur, ni la largeur ni la position de l'objet ne varient
Redim horizontal déplacer (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 "Propriétés des objets" :

Constante (valeur)	Commentaire
Redim vertical agrandir (1)	Si la fenêtre s'agrandit de 50% en hauteur, l'objet s'allonge de 50% vers le bas
Redim vertical aucun (0)	Si la fenêtre s'agrandit en hauteur, ni la hauteur ni la position de l'objet ne varient
Redim vertical déplacer (2)	Si la fenêtre s'agrandit de 100 pixels en hauteur, l'objet se déplace de 100 pixels vers le bas

Référence : OBJET FIXER REDIMENSIONNEMENT

OBJET LIRE TAILLE SOUS FORMULAIRE

OBJET LIRE TAILLE SOUS FORMULAIRE(largeur; hauteur)

Paramètres	Type		Description
largeur	Entier long	←	Largeur de l'objet sous-formulaire
hauteur	Entier long	←	Hauteur de l'objet sous-formulaire

La nouvelle 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 déplacés et/ou redimensionnés en fonction des caractéristiques de l'objet sous-formulaire lui-même. Dans l'événement formulaire Sur chargement ou Sur redimensionnement, le sous-formulaire peut appeler cette commande pour calculer la place dont il dispose afin d'afficher son contenu.

- 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*.

Référence : OBJET FIXER SOUS FORMULAIRE, OBJET LIRE SOUS FORMULAIRE

OBJET LIRE SOUS FORMULAIRE

OBJET LIRE SOUS FORMULAIRE({*; }objet; ptrTable; sousFormDetail{; sousFormListe})

Paramètres	Type		Description
*		→	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)

La nouvelle 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.

Référence : OBJET FIXER SOUS FORMULAIRE, OBJET LIRE TAILLE SOUS FORMULAIRE

Commandes renommées

En raison des nouvelles commandes v13, les commandes existantes suivantes ont été renommées :

Nom versions précédentes	*Nouveau nom v13*
OBJET FIXER ALIGNEMENT	**OBJET FIXER ALIGNEMENT HORIZONTAL**
OBJET Lire alignement	**OBJET Lire alignement horizontal**

Note de compatibilité Dans 4D v13, le thème de constantes "Alignement des objets" n'existe plus, son contenu a été fusionné avec le nouveau thème "Propriétés des objets".

Recherches et tris

LIRE DESTINATION RECHERCHE

LIRE DESTINATION RECHERCHE(destinationType; destinationObjet{; destinationPtr})

Paramètres	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

La nouvelle 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 existante FIXER DESTINATION RECHERCHE (modifiée en v13, voir ci-dessous).

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".
Le tableau suivant décrit les combinaisons possibles :

destinationType (valeur)	Commentaire
Vers sélection courante (0)	*destinationObjet* est une chaîne vide
Vers ensemble (1)	*destinationObjet* contient le nom de l'ensemble
Vers sélection temporaire (2)	*destinationObjet* contient le nom de la sélection temporaire
Vers variable (3)	*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.

- 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
```

Référence : FIXER DESTINATION RECHERCHE

Lire limite recherche

Lire limite recherche → Entier long

Paramètres	Type		Description
Résultat	Entier long	←	Nombre limite d'enregistrements, 0 = nombre illimité

La nouvelle 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.

FIXER DESTINATION RECHERCHE

FIXER DESTINATION RECHERCHE(destinationType; {destinationObjet{; *destinationPtr*}})

Paramètres	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 *ou Chaîne vide*
destinationPtr	*Pointeur*	→	*Pointeur vers la variable locale si destinationType=3*

La commande FIXER DESTINATION RECHERCHE admet un nouveau paramètre, permettant d'utiliser une syntaxe alternative lors de la définition d'une variable en destination de recherche :

FIXER DESTINATION RECHERCHE(Vers variable;""; ->$MaVar)

Serveur Web

Plusieurs nouvelles commandes permettent de gérer les nouvelles fonctions proposées par le serveur Web de 4D, notamment la gestion des sessions et des cookies. Ces nouvelles fonctions sont décrites dans la section chapitre 4, "Serveur Web", page 67.

WEB FERMER SESSION

WEB FERMER SESSION(idSession)

Paramètres	Type		Description
idSession	Texte	→	UUID de session

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 session 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.

WEB FIXER OPTION

WEB FIXER OPTION(sélecteur ; valeur)

Paramètres	Type		Description
sélecteur	Entier long	→	Code de l'option à modifier
valeur	Entier long, Texte	→	Valeur de l'option

La nouvelle 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 nouveau thème "Serveur Web" et dans *valeur* la nouvelle valeur de l'option :

Note Dans 4D v13, le thème "Serveur Web" regroupe toutes les constantes de gestion du serveur Web, y compris celles auparavant disponibles via le thème "Paramètres de la base". Il est désormais recommandé d'utiliser exclusivement la commande WEB FIXER OPTION pour définir les options Web (cf. également FIXER PARAMETRE BASE, Lire parametre base).

sélecteur (*valeur*)	Description
Web Adresse IP d'écoute (16)	*identique "Paramètres de la base"*
Web Enreg requêtes (29)	*identique "Paramètres de la base"*
Web Jeu de caractères (17)	*identique "Paramètres de la base"*
Web Niveau de compression HTTP (50)	*identique "Paramètres de la base"*
Web Numéro de port HTTPS (39)	*identique "Paramètres de la base"*
Web Process Web simultanés maxi (18)	*identique "Paramètres de la base"*
Web Seuil de compression HTTP (51)	*identique "Paramètres de la base"*
Web Taille max requêtes (27)	*identique "Paramètres de la base"*

Web Conserver les sessions (70)	Permet d'activer ou d'inactiver le nouveau mode de gestion des sessions. **Valeurs possibles** : 1 (activer mode) ou 0 (inactiver mode) **Valeur par défaut** : 1 pour les bases créées en v13, 0 pour les bases converties A noter que ce mode active également le mécanisme de réutilisation des contextes temporaires. Pour plus d'informations sur ce mécanisme, reportez-vous à la description de cette option dans les Propriétés de la base.
Web Nom du cookie de session (73)	Permet de définir le nom du cookie utilisé pour stocker l'ID de session. **Valeurs possibles** : Texte **Valeur par défaut** : "4DSID" (passez une chaîne vide pour rétablir la valeur par défaut)
Web Nombre de sessions max (71)	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 session Web est appelée) si le serveur Web a besoin d'en créer une nouvelle. **Valeurs possibles** : 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 Timeout session (72)	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 possibles** : Entier long (minutes) **Valeur par défaut** : 480 minutes (passez 0 pour rétablir la valeur par défaut)
Web Timeout process (78)	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 session Web est appelée puis le contexte de la session est détruit. **Valeurs possibles** : Entier long (minutes) **Valeur par défaut** : 480 minutes (passez 0 pour rétablir la valeur par défaut)

La portée des constantes provenant du thème "Paramètres de la base" est inchangée par rapport aux versions précédentes de 4D.

Les nouvelles options s'appliquent au serveur Web local : si elles sont appelées sur un 4D en mode distant, elles s'applique uniquement au serveur Web de ce 4D distant (et non au serveur Web de 4D Server ou des autres 4D distants). Les paramétrages effectués ne sont pas stockés. En revanche, ils restent valides durant toute la session 4D, même si le serveur HTTP est redémarré.

Référence : WEB LIRE OPTION

WEB LIRE EXPIRATION SESSION

WEB LIRE EXPIRATION SESSION(idSession ; dateExp ; heureExp) →

Paramètres	Type		Description
idSession	Texte	→	UUID de session
dateExp	Date	←	Date d'expiration du cookie
heureExp	Heure	←	Heure d'expiration du cookie

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 Web Timeout session. Par exemple :
Première requête, Lundi à 01h00
-> envoi du cookie 4DSID xxxyyy expiration I+24h = Mardi 01:00
Deuxième requête, Lundi à 01h10
-> envoi du cookie 4DSID xxxyyy expiration I+24h = Mardi 01:10
Troisième requête, Mardi à 04h00 : cookie expiré
-> envoi du cookie 4DSID aaabbb expiration I+24h = Mercredi 01:00

Référence : WEB FIXER OPTION

WEB Lire ID session courante

WEB Lire ID session courante → Texte

Paramètres	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Texte	←	UUID de la session

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 → Entier long

Paramètres	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	Nombre de parties du corps

La commande WEB Lire nombre parties corps retourne le nombre de parties qui composent le body (corps) reçu.

Référence : WEB LIRE PARTIE CORPS

WEB LIRE OPTION

WEB LIRE OPTION(sélecteur ; valeur)

Paramètres	Type		Description
sélecteur	Entier long	→	Code de l'option à modifier
valeur	Entier long, Texte	←	Valeur de l'option

La commande WEB LIRE OPTION permet de lire la valeur courante d'une option de fonctionnement du serveur Web de 4D.

Le paramètre *sélecteur* l'option Web à lire. Passez dans ce paramètre une constante du thème "Serveur Web". Pour plus d'informations, reportez-vous à la description de la commande WEB FIXER OPTION.

Référence : WEB FIXER OPTION

WEB LIRE PARTIE CORPS

WEB LIRE PARTIE CORPS(partie ; contPartie ; nomPartie{ ; typeMime ; nomFichier})

Paramètres	Type		Description
partie	Entier long	→	Numéro de partie
contPartie	BLOB, Texte	←	Contenu de la partie
nomPartie	Texte	←	Nom de la variable "input"
typeMime	Texte	←	Type mime du fichier posté
nomFichier	Texte	←	Nom du fichier posté

Cette commande, 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.

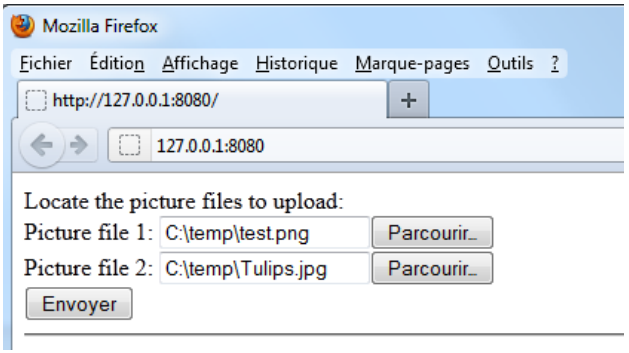
Le paramètre *contPartie* récupère le contenu de la partie. Lorsque les parties à récupérer sont des fichiers, vous devez passer un paramètre de type BLOB. Dans le cas de variables TEXT postées dans un formulaire Web, vous pouvez passer un paramètre de type texte.

Le paramètre *nomPartie* récupère le nom de la variable du champ input.

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 afin d'obtenir directement la position des parties du formulaire.

- 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)
$vDestinationFolder:=Dossier 4D(Dossier racine HTML)+"photos"+Séparateur dossier //dossier "WebFolder/photos"
LISTE DES DOCUMENTS($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
```

Référence : WEB Lire nombre parties corps

Renommage des commandes du thème Serveur Web

Pour des raisons de cohérence, les commandes existantes du thème serveur Web (ainsi que les nouvelles commandes) ont été préfixées "WEB" dans 4D v13 :

Noms 4D v13	Anciens noms (4D v12.x)
WEB ARRETER SERVEUR	ARRETER SERVEUR WEB
WEB Connexion securisee	Connexion Web securisee
WEB DEMARRER SERVEUR	LANCER SERVEUR WEB
WEB ENVOYER BLOB	ENVOYER BLOB HTML
WEB ENVOYER DONNEES	ENVOYER DONNEES HTTP
WEB ENVOYER FICHIER	ENVOYER FICHIER HTML
WEB ENVOYER REDIRECTION HTTP	ENVOYER REDIRECTION HTTP
WEB ENVOYER TEXTE	ENVOYER TEXTE HTML
WEB FERMER SESSION	*nouvelle commande v13*
WEB FIXER ENTETE HTTP	FIXER ENTETE HTTP
WEB FIXER OPTION	*nouvelle commande v13*
WEB FIXER PAGE ACCUEIL	FIXER PAGE ACCUEIL
WEB FIXER RACINE	FIXER RACINE HTML
WEB LIRE CORPS HTTP	LIRE CORPS HTTP
WEB LIRE ENTETE HTTP	LIRE ENTETE HTTP
WEB LIRE EXPIRATION SESSION	*nouvelle commande v13*
WEB Lire ID Session courante	*nouvelle commande v13*
WEB Lire nombre parties corps	*nouvelle commande v13*
WEB LIRE OPTION	*nouvelle commande v13*

WEB LIRE PARTIES CORPS	*nouvelle commande v13*
WEB LIRE STATISTIQUES	STATISTIQUES DU CACHE WEB
WEB LIRE VARIABLES	LIRE VARIABLES FORMULAIRE WEB
WEB Valider digest	Valider mot de passe digest Web

Notes - Les commandes obsolètes Contexte Web, FIXER LIMITES AFFICHAGE WEB, FIXER TEMPORISATION WEB et FIXER EXECUTABLE CGI ont été préfixées "_o_" et sont désormais désactivées. Leur exécution génère l'erreur 33 (*Méthode ou fonction non implémentée*).
- La commande TRAITER BALISES HTML a été renommée TRAITER BALISES 4D et transférée dans le thème "Outils" (son fonctionnement est inchangé).

Tableaux

SELECTION LIMITEE VERS TABLEAU

SELECTION LIMITEE VERS TABLEAU (début; fin*{*; leChamp | laTable ; tableau*}* {; leChamp2 ; tableau2 ; ... ; leChampN ; tableauN})

Il est désormais 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 nouveau paramètre *.

- Utilisation des 50 premiers enregistrements courants de la table [Factures] pour charger divers tableaux :

```
  // 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)
```

Référence : SELECTION VERS TABLEAU

SELECTION VERS TABLEAU

SELECTION VERS TABLEAU *{(leChamp | laTable ; tableau {; leChamp2 ; tableau2 ; ... ; leChampN ; tableauN} {; *})}*

Paramètres	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*

La commande SELECTION VERS TABLEAU accepte désormais un * en dernier paramètre. Lorsque vous passez ce 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 désormais être appelée sans aucun paramètre.

Les types des tableaux sont vérifiés au moment de l'exécution de la ligne finale (sans 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 ou de construire un tableau dans une boucle.

- Dans 4D v13, la ligne :

SELECTION VERS TABLEAU ([Clients]Nom;tabClients;[Clients]ID;tabID;[Société]Nom;tabSoc)

... peut s'écrire :

SELECTION VERS TABLEAU ([Clients]Nom;tabClients;*)
SELECTION VERS TABLEAU ([[Clients]ID;tabID;*)
SELECTION VERS TABLEAU ([Société]Nom;tabSoc)

Référence : TABLEAU VERS SELECTION, SELECTION LIMITEE VERS TABLEAU

TABLEAU VERS SELECTION

TABLEAU VERS SELECTION*{*(tableau ;leChamp{; tableauN; leChampN} *{; *})}*

Paramètres	Type		Description
tableau	Tableau	→	Tableau à copier dans la sélection
leChamp	Champ	←	Champ recevant les valeurs du tableau
*	*Opérateur*	→	*Attente d'exécution*

La commande TABLEAU VERS SELECTION accepte désormais un * en dernier paramètre. Lorsque vous passez ce 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 désormais 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.

- 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
```

Référence : SELECTION VERS TABLEAU

TEXTE VERS TABLEAU

TEXTE VERS TABLEAU(varTexte; tabTexte; largeur; nomPolice; taillePolice{; stylePolice{; *}})

Paramètres	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

La nouvelle 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". 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.

- 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
```

- 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
```

- 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)
```

- 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)
```

Autres modifications

Pour des raisons de clarté ou de cohérence, diverses modifications sémantiques et d'organisation ont été apportées au langage de 4D v13.

Triggers

- La commande Evenement moteur est renommée **Evenement trigger** dans 4D v13.
- Le thème de constantes "Evénements moteur" est également renommé "Evénements trigger".

Le fonctionnement de la commande et des constantes est inchangé.

Ces renommages permettent de mieux différencier les événements triggers des nouveaux événements système introduits dans 4D v13 (cf. paragraphe "Sur événement système", page 171).

Web Services

Les noms des commandes associées aux Web services ont été modifiés.

Web Services (Client)

Pour plus de cohérence, les commandes du thème "Web Services (Client)" ont été renommées dans 4D v13 :

Nouveaux noms 4D v13	Anciens noms (4D v12.x)
WEB SERVICE APPELER	APPELER WEB SERVICE
WEB SERVICE AUTHENTIFIER	AUTHENTIFIER WEB SERVICE
WEB SERVICE FIXER OPTION	FIXER OPTION WEB SERVICE
WEB SERVICE FIXER PARAMETRE	FIXER PARAMETRE WEB SERVICE
WEB SERVICE Lire infos	Lire infos erreur Web Service
WEB SERVICE LIRE RESULTAT	LIRE RESULTAT WEB SERVICE

Le fonctionnement de ces commandes est inchangé.

Web Services (Serveur)

Les commandes du thème "Web Services (Serveur)" ont également été préfixées dans 4D v13 :

Nouveaux noms 4D v13	Anciens noms (4D v12.x)
SOAP DECLARATION	DECLARATION SOAP
SOAP ENVOYER ERREUR	ENVOYER ERREUR SOAP
SOAP Requete	Est une requete SOAP
SOAP Lire information	Lire informations SOAP

Le fonctionnement de ces commandes est inchangé.

Constantes renommées

Les constantes suivantes du thème "Type du process" ont été renommées :

Nouveaux noms 4D v13	Anciens noms (4D v12.x)
_o_Process Web avec contexte	Process Web avec contexte
Process Web 4D distant	Process Web 4D Client

La constante suivante du thème "Documents système" a été renommée :

Nouveau nom 4D v13	Ancien nom (4D v12.x)
Est un dossier	Est un répertoire

La constante suivante du thème "Web Services (Client)" a été renommée :

Nouveau nom 4D v13	Ancien nom (4D v12.x)
Web service code statut HTTP	Web service code erreur HTTP

Commandes déplacées

Pour plus de clarté, des commandes ont changé de thème dans 4D v13 :

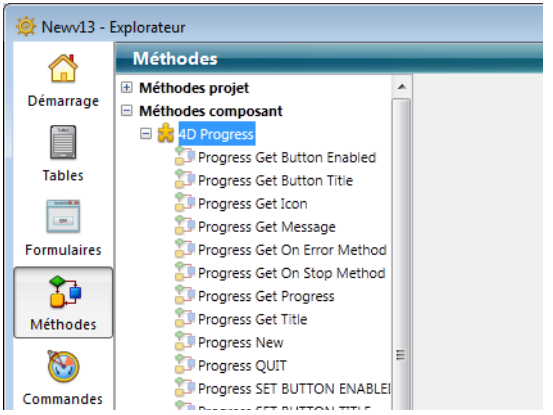
- Lire texte edite passe du thème "Evenement formulaire" au thème "Gestion de la saisie".
- SELECTIONNER TEXTE et TEXTE SELECTIONNE passent du thème "Interface utilisateur" au thème "Gestion de la saisie".
- NE PAS VALIDER et VALIDER passent du thème "Gestion de la saisie" au thème "Saisie".

4D Progress

4D v13 contient un nouveau composant intégré nommé 4D Progress. Ce composant vous permettra d'ouvrir une ou plusieurs barres de progression dans une même fenêtre (à l'image de l'interface du *Finder* de Mac OS).

Chaque barre de progression se voit attribuer un ID, généré automatiquement par la méthode Progress New, qui sera utilisé par toutes les méthodes projet du composant pour gérer les propriétés et les valeurs dans les boîtes de dialogue de progression.

Les méthodes projet du composant sont listées dans la page **Méthodes composant** de l'Explorateur :



Progress Get Button Enabled

Progress Get Button Enabled (id) → Booléen

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
Résultat	Booléen	←	Vrai = bouton Stop affiché, sinon Faux

La méthode Progress Get Button Enabled retourne Vrai si la barre de progression désignée par le paramètre *id* affiche un bouton **Stop**. Si la barre n'affiche pas de bouton **Stop** (fonctionnement par défaut), la méthode retourne Faux.

Référence : Progress SET BUTTON ENABLED

Progress Get Button Title

Progress Get Button Title (id) → Texte

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
Résultat	Texte	←	Libellé du bouton Stop

Note Cette méthode est utilisable sous Windows uniquement. Sous Mac OS, le bouton d'arrêt n'a pas de libellé.

La méthode Progress Get Button Title retourne le libellé courant du bouton d'arrêt de la barre de progression désignée par le paramètre *id*.

Par défaut, le libellé est "Stop". A noter que la méthode retourne le libellé courant même si le bouton **Stop** n'est pas affiché.

Référence : Progress SET BUTTON TITLE

Progress Get Icon

Progress Get Icon (id) → Image

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
Résultat	Image	←	Icône de la barre de progression

La méthode Progress Get Icon retourne l'icône de la barre de progression désignée par le paramètre *id*.

Référence : Progress SET ICON

Progress Get Message

Progress Get Message (id) → Texte

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
Résultat	Texte	←	Message de la barre de progression

La méthode Progress Get Message retourne le message de la barre de progression désignée par le paramètre *id*.

Référence : Progress SET MESSAGE

Progress Get On Error Method

Progress Get On Error Method → Texte

Paramètres	Type		Description
Résultat	Texte	←	Nom de la méthode appelée en cas d'erreur (si définie)

La méthode Progress Get On Error Method retourne le nom de la méthode projet de la base hôte appelée en cas d'erreur lors de l'utilisation d'une barre de progression.

Si aucune méthode d'erreur n'est définie, la méthode retourne une chaîne vide.

Référence : Progress SET ON ERROR METHOD

Progress Get On Stop Method

Progress Get On Stop Method(id) → Texte

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
Résultat	Texte	←	Nom de la méthode appelée en cas de clic sur le bouton Stop (si définie)

La méthode Progress Get On Stop Method retourne le nom de la méthode projet de la base hôte appelée lorsque l'utilisateur clique sur le bouton **Stop** de la barre de progression désignée par le paramètre *id*.

Si aucune méthode n'est associée au **Stop**, la méthode retourne une chaîne vide.

Référence : Progress SET ON STOP METHOD

Progress Get Progress

Progress Get Progress (id) → Réel

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
Résultat	Réel	←	Valeur de la barre de progression

La méthode Progress Get Progress retourne la valeur courante associée à la barre de progression désignée par le paramètre *id*.

Référence : Progress SET PROGRESS

Progress Get Title

Progress Get Title (id) → Texte

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
Résultat	Texte	←	Titre de la barre de progression

La méthode Progress Get Title retourne le titre principal de la barre de progression désignée par le paramètre *id*.

Référence : Progress SET TITLE

Progress New

Progress New → Entier long

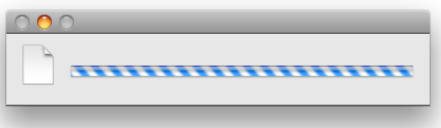
Paramètres	Type		Description
Résultat	Entier long	←	ID de la nouvelle barre de progression

La méthode Progress New crée une nouvelle barre de progression et retourne son numéro d'ID. Ce numéro est unique pendant la durée de vie de la barre de progression mais pourra être réutilisé.

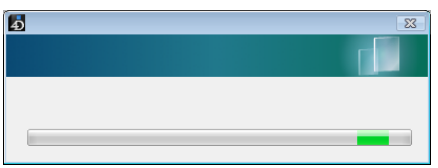
Au premier appel de cette méthode, un process local est créé et une nouvelle fenêtre centrée est ouverte au-dessus de la fenêtre principale. Par défaut, cette fenêtre :

- contient une barre de progression indéfinie
- n'a pas de message.

Mac OS



Windows



Progress QUIT

Progress QUIT (id)

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression

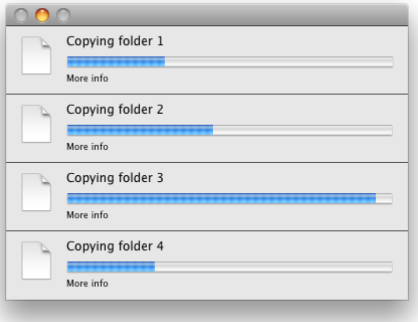
La méthode Progress QUIT vous permet de fermer la barre de progression référencée par le paramètre *id*.

Si *id* désigne la seule barre de progression affichée, la fenêtre de progression est également refermée (ainsi que le process local). Sinon, la fenêtre est redimensionnée afin de ne contenir que les barres ouvertes.

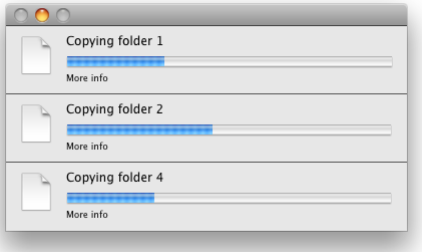
- Si la barre nommée "Copying folder 3" a le numéro d'ID 3 :

Progress QUIT(3)

Avant

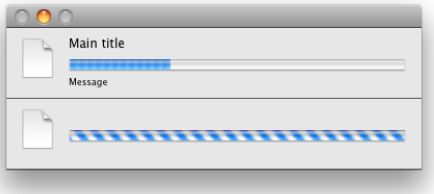


Après

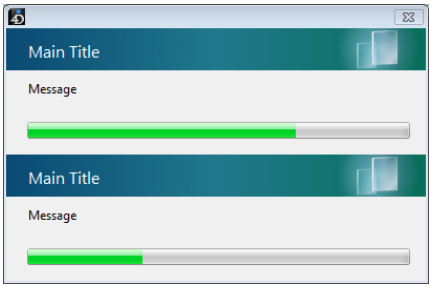


Si une fenêtre de progression était déjà ouverte à l'appel de cette méthode, elle est redimensionnée afin d'afficher une nouvelle barre de progression sous la ou les précédente(s), dans le même process :

Mac OS



Windows



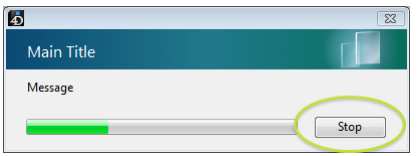
Progress SET BUTTON ENABLED

Progress SET BUTTON ENABLED (id; bouton)

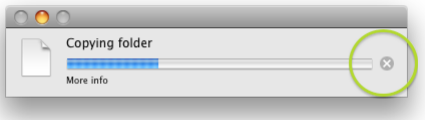
Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
bouton	Booléen	→	Vrai = Afficher, Faux = Masquer

La méthode Progress SET BUTTON ENABLED vous permet d'ajouter un bouton **Stop** à la barre de progression désignée par le paramètre *id*.

Par défaut, les barres de progression n'ont pas de bouton **Stop**. Si vous passez Vrai dans le paramètre *bouton*, un bouton sera affiché :



Windows



Mac OS

Vous pouvez gérer l'effet du clic sur le bouton **Stop** à l'aide de la méthode Progress SET ON STOP METHOD ou en testant la valeur de la méthode Progress Stopped.

Référence : Progress Get Button Enabled

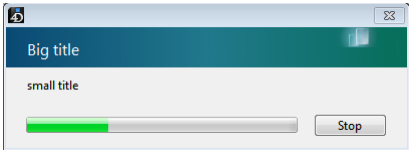
Progress SET BUTTON TITLE

Progress SET BUTTON TITLE (id; titre)

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
titre	Texte	→	Libellé du bouton Stop (Windows)

Note Cette méthode est utilisable sous Windows uniquement. Sous Mac OS, le bouton d'arrêt n'a pas de libellé.

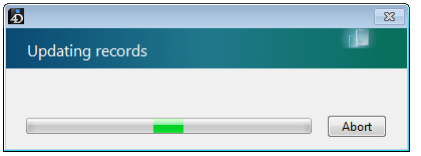
La méthode Progress SET BUTTON TITLE vous permet de modifier le libellé du bouton **Stop** de la barre de progression désignée par le paramètre *id*. Par défaut, le libellé du bouton d'arrêt est "Stop" :



Note Par défaut, les barres de progression n'ont pas de bouton **Stop**. Pour qu'une barre de progression contienne un bouton **Stop**, vous devez utiliser la méthode Progress SET BUTTON ENABLED.

- Vous souhaitez utiliser le libellé "Abort" :

```
<>ID:=Progress New
...
Progress SET BUTTON TITLE (<>ID;"Abort")
```



Référence : Progress Get Button Title

Progress SET FONT SIZES

Progress SET FONT SIZES (tailleTitres{; tailleMessages{; tailleBoutons}})

Paramètres	Type		Description
tailleTitres	Texte	→	Taille de police pour les titres
tailleMessages	Texte	→	Taille de police pour les messages
tailleBoutons	Texte	→	(Windows) Taille de police pour les boutons Stop

La méthode Progress SET FONT SIZES vous permet de modifier la taille des polices de caractères utilisées pour les différents textes affichés dans toutes les fenêtres de progression.

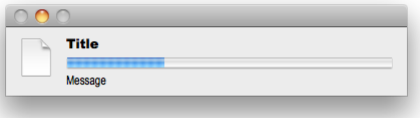
Passez dans les paramètres *tailleTitres*, *tailleMessages* et *tailleBoutons* les tailles des polices de caractères à utiliser. Si vous ne souhaitez pas modifier une taille, passez -1 dans le paramètre correspondant.

- On souhaite changer uniquement la taille des messages :

```
Progress SET FONT SIZES(-1; 13)
```

- On change la police et la taille des titres et des messages :

Progress SET FONTS("Arial Black"; "Arial narrow")
Progress SET FONT SIZES(13; 12)



Référence : Progress SET FONTS

Progress SET FONTS

Progress SET FONTS (policeTitres{; policeMessages{; policeBoutons}})

Paramètres	Type		Description
policeTitres	Texte	→	Police à utiliser pour les titres
policeMessages	Texte	→	Police à utiliser pour les messages
policeBoutons	Texte	→	(Windows) Police à utiliser pour les boutons Stop

La méthode Progress SET FONTS vous permet de modifier les polices de caractères utilisées pour les différents textes affichés dans toutes les fenêtres de progression.

Passez dans les paramètres *policeTitres*, *policeMessages* et *policeBoutons* les noms des polices de caractères à utiliser. Si vous ne souhaitez pas modifier une police, passez une chaîne vide ("") dans le paramètre correspondant.

- On souhaite changer uniquement la police des messages :

Progress SET FONTS(""; "Arial")

Référence : Progress SET FONT SIZES

Progress SET ICON

Progress SET ICON (id; icône{; premierPlan})

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
icône	Image	→	Image à afficher en tant qu'icône
premierPlan	Booléen	→	Montrer la progression au premier plan

La méthode Progress SET ICON vous permet de modifier l'icône affichée dans la barre de progression. Par défaut, les icônes suivantes sont affichées :

Windows

Mac OS

Passez dans *id* le numéro d'ID unique de la barre de progression, retourné par la méthode Progress New.

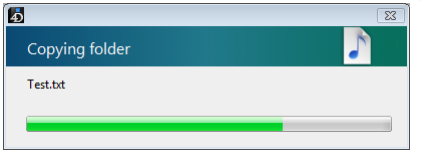
Passez dans *icône* l'image (variable ou champ) à utiliser comme icône dans la fenêtre de la barre de progression. La taille maximale de cette icône doit être :

- sous Mac OS, 40 x 40 pixels
- sous Windows, 40 x 80 pixels

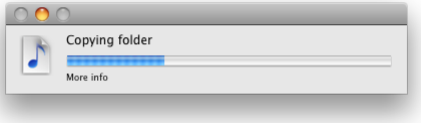
Si vous passez une icône de taille inférieure à ces limites, elle sera centrée et non redimensionnée.
Si elle est de taille supérieure, elle sera centrée et redimensionnée.

Passez Vrai dans *premierPlan* si vous voulez forcer le passage de la fenêtre de progression au premier plan de l'application.

Exemples d'icônes personnalisées :



Windows



Mac OS

Référence : Progress Get Icon

Progress SET MESSAGE

Progress SET MESSAGE (id; message{; premierPlan}}})

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
message	Texte	→	Message de la barre
premierPlan	Booléen	→	Montrer la progression au premier plan

La méthode Progress SET MESSAGE vous permet de modifier le message affiché dans la barre de progression.

Passez dans *id* le numéro d'ID unique de la barre de progression, retourné par la méthode Progress New.

Passez dans *message* le texte à modifier sous le titre principal (Windows) ou sous la barre de progression (Mac OS).

Passez Vrai dans *premierPlan* si vous voulez forcer le passage de la fenêtre de progression au premier plan de l'application.

Référence : Progress Get Message

Progress SET ON ERROR METHOD

Progress SET ON ERROR METHOD (nomMéthode)

Paramètres	Type		Description
nomMéthode	Texte	→	Nom de la méthode d'erreur

La méthode Progress SET ON ERROR METHOD vous permet de désigner une méthode à exécuter en cas d'erreur lors de l'utilisation des barres de progression (par exemple *id* inconnu, nombre de paramètres incorrect, etc.).

Passez dans *nomMéthode* le nom de la méthode projet de la base hôte à appeler en cas d'erreur. Cette méthode sera commune à toutes les fenêtres de progression de l'application.

Note Attention, comme la méthode Progress SET ON ERROR METHOD appartient à un composant, vous devez veiller à affecter la propriété "Partagée entre composants et base hôte" à la méthode *nomMéthode*, sinon une erreur sera retournée.

La méthode *nomMéthode* sera appelée avec trois paramètres :

- $1 (Entier long) : numéro de l'erreur
- $2 (Texte) : texte de l'erreur
- $3 (Entier long) : ID unique de la barre de progression

- Voici un exemple de méthode appelée en cas d'erreur. Cette méthode a été déclarée "partagée" en mode Développement :

```
C_ENTIER LONG($1)
C_TEXTE($2)
C_ENTIER LONG($3)

C_ENTIER LONG($ErrorID)
C_TEXTE($ErrorText)
C_ENTIER LONG($ProgressID)

$ErrorID:=$1
$ErrorText:=$2
$ProgressID:=$3

$Error:=""
$Error:=$Error+"Erreur numéro : "+Chaine($ErrorID)+Caractere(Retour chariot)
$Error:=$Error+$ErrorText+Caractere(Retour chariot)
$Error:=$Error+"ID progression : "+Chaine($ProgressID)
ALERTE($Error)
```

Référence : Progress Get On Error Method

Progress SET ON STOP METHOD

Progress SET ON STOP METHOD (id; nomMéthode)

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
nomMéthode	Texte	→	Nom de la méthode de stop

La méthode Progress SET ON STOP METHOD vous permet de désigner une méthode à exécuter lorsque l'utilisateur clique sur le bouton **Stop** de la barre de progression.

Par défaut, une barre de progression ne contient pas de bouton **Stop**. Vous devez l'afficher explicitement à l'aide de la méthode Progress SET BUTTON ENABLED.

Passez dans *id* le numéro d'ID unique de la barre de progression, retourné par la méthode Progress New.

Passez dans *nomMéthode* le nom de la méthode projet de la base hôte à appeler lors du clic sur le bouton **Stop**. Cette méthode recevra l'ID unique de la barre de progression en premier paramètre. Elle sera exécutée dans un nouveau process lancé par le composant.

Note Attention, comme la méthode Progress SET ON STOP METHOD appartient à un composant, vous devez veiller à affecter la propriété "Partagée entre composants et base hôte" à la méthode *nomMéthode*, sinon une erreur sera retournée.

Référence : Progress Get On Stop Method

Progress SET PROGRESS

Progress SET PROGRESS (id; progression{; message{; premierPlan}}})

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
progression	Réel	→	Valeur de progression ([0...1] ou -1)
message	Texte	→	Message de la barre
premierPlan	Booléen	→	Montrer la progression au premier plan

La méthode Progress SET PROGRESS vous permet de modifier la valeur de la barre de progression ainsi que les informations affichées dans la fenêtre de progression. Cette méthode est utile pour la mise à jour d'une barre de progression au sein d'une boucle.

Passez dans *id* le numéro d'ID unique de la barre de progression, retourné par la méthode Progress New.

Passez dans *progression* la valeur courante de la barre de progression. Vous pouvez passer une valeur réelle (entre 0 et 1) ou -1 afin de définir une barre de progression indéfinie (aussi appelée "Barber shop" sous Mac OS).

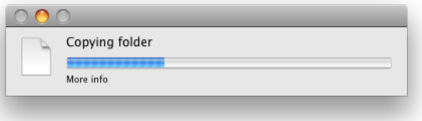
Passez dans *message* un texte additionnel à afficher sous le titre principal (Windows) ou sous la barre de progression (Mac OS). Ce paramètre est optionnel.

Passez Vrai dans *premierPlan* si vous voulez forcer le passage de la fenêtre de progression au premier plan de l'application.

- Mise à jour de la progression de la barre :

```
$P:=Progress New // on crée une nouvelle barre
  // Effectuer un traitement dans une boucle
Boucle($i;1;100)
     // ... code du traitement
     // Mise à jour de la barre de progression
  $r:=$i/100
  Progress SET PROGRESS ($P;$r;"More info")
Fin de boucle
```

```
// Suppression de la barre une fois le traitement terminé
PROGRESS DELETE($P)
```



Référence : Progress Get Progress

Progress SET TITLE

Progress SET TITLE (id; titre{; progression{; message{; premierPlan}}})

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
titre	Texte	→	Titre de la barre
progression	Réel	→	Valeur de progression ([0...1] ou -1)
message	Texte	→	Message de la barre
premierPlan	Booléen	→	Montrer la progression au premier plan

La méthode Progress SET TITLE vous permet de définir le titre de la barre de progression ainsi que les informations affichées dans la fenêtre de progression.

Passez dans *id* le numéro d'ID unique de la barre de progression, retourné par la méthode Progress New.

Passez dans *titre* le texte principal à afficher dans la fenêtre de progression.

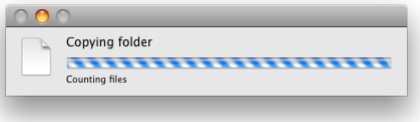
Passez dans *progression* la valeur courante de la barre de progression (facultatif). Vous pouvez passer une valeur réelle (entre 0 et 1) ou -1 afin de définir une barre de progression indéfinie (aussi appelée "Barber shop" sous Mac OS).

Passez dans *message* un texte additionnel à afficher sous le titre principal (Windows) ou sous la barre de progression (Mac OS). Ce paramètre est optionnel.

Passez Vrai dans *premierPlan* si vous voulez forcer le passage de la fenêtre de progression au premier plan de l'application.

- Création d'une fenêtre de progression simple :

```
$P:=Progress New
Progress SET TITLE($P;"Copying folder";-1;"Counting files")
```



Référence : Progress Get Title

Progress SET WINDOW VISIBLE

Progress SET WINDOW VISIBLE (visible{; posHor; posVert{; premierPlan}})

Paramètres	Type		Description
visible	Booléen	→	Vrai = Montrer, Faux = Cacher
posHor	Entier long	→	Coordonnée gauche de la fenêtre -1 = Pas de changement
posVert	Entier long	→	Coordonnée haute de la fenêtre -1 = Pas de changement
premierPlan	Booléen	→	Montrer la progression au premier plan

La méthode Progress SET WINDOW VISIBLE vous permet de gérer l'affichage de la fenêtre de progression, si elle existe.

Le paramètre *visible* indique si la fenêtre doit être visible ou non (par défaut, la fenêtre est visible). Passez Faux dans ce paramètre pour masquer la fenêtre. Passez Vrai pour l'afficher de nouveau si elle existe.

Les paramètres *posHor* et *posVert* permettent de modifier l'emplacement de la fenêtre de progression à l'écran. Vous pouvez passer dans ces paramètres des valeurs représentant le décalage en pixels de la fenêtre vers la droite (*posHor*) ou vers le bas (*posVert*) par rapport aux coordonnées de la fenêtre principale de l'application (sous Windows) ou de l'écran (sous Mac OS).
Pour ne pas modifier les coordonnées (si vous souhaitez utiliser le paramètre *premierPlan* sans déplacer la fenêtre), passez -1 dans chacun de ces paramètres.

Passez Vrai dans *premierPlan* si vous voulez forcer le passage de la fenêtre de progression au premier plan de l'application.

- Placer la fenêtre de progression de 50 pixels du bord gauche et 100 pixels du bord haut :

  ***Progress SET WINDOW VISIBLE*(Vrai; 50; 100)**

- Masquer la fenêtre de progression :

  ***Progress SET WINDOW VISIBLE*(Faux)**

- Afficher la fenêtre de progression et la faire passer au premier plan sans modifier sa position :

  ***Progress SET WINDOW VISIBLE*(Vrai; -1; -1; Vrai)**

Progress Stopped

Progress Stopped (id) → Booléen

Paramètres	Type		Description
id	Entier long	→	ID de la barre de progression
Résultat	Booléen	←	Vrai = L'utilisateur a cliqué sur le bouton Stop

La méthode Progress Stopped retourne Vrai si l'utilisateur a cliqué sur le bouton **Stop** de la barre de progression désignée par le paramètre *id*.

Vous devez appeler cette méthode pour tester si l'utilisateur a cliqué sur le bouton **Stop**. Le bouton ne déclenche pas d'événement lui-même.

- Exemple de barre de progression sur une boucle :

```
$ProgressID:=Progress New // création d'une nouvelle barre
  // La barre doit avoir un bouton Stop
Progress SET BUTTON ENABLED ($ProgressID;Vrai)
Boucle ($i;1;100)
     // Tant que la progression n'est pas stoppée...
  Si (Non(Progress Stopped ($ProgressID)))
     Progress SET TITLE ($ProgressID;"Test progression #"+Chaine($Pro-
gressID))
     Progress SET PROGRESS ($ProgressID;$i/100)
     Progress SET MESSAGE ($ProgressID;Chaine(100*$i/100)+" %")
       (...)
  Sinon // L'utilisateur a cliqué sur Stop
     $i:=100 // On sort de la boucle
  Fin de si
Fin de boucle
  // Fermeture finale de la barre (le bouton Stop lui-même ne fait rien)
Progress QUIT ($ProgressID)
```

4D SVG

Le composant 4D SVG intégré dans 4D v13 a été mis à jour. De nouvelles méthodes ont été ajoutées et des méthodes existantes ont été modifiées.

Nouvelles méthodes

SVG_ADD_NAMESPACE

SVG_ADD_NAMESPACE (objetSVG ; préfixe {; URI })

Paramètres	Type		Description
objetSVG	Ref_SVG	→	Référence d'objet SVG
préfixe	Texte	→	Préfixe du namespace$
URI	Texte	→	URI du namespace

La méthode SVG_ADD_NAMESPACE ajoute un attribut XML namespace à la racine de l'arbre DOM de la structure SVG référencée par *objetSVG*. Cette méthode vous permet notamment d'ajouter un namespace à un extrait de code SVG.

Passez dans le paramètre *préfixe* une chaîne contenant le préfixe de l'attribut namespace. Vous pouvez utiliser l'une des constantes suivantes :

- "svgNS" pour le namespace SVG standard (http://www.w3.org/2000/svg)
- "xlinkNS" pour le namespace standard XLink (http://www.w3.org/1999/xlink)

Dans ce cas, le paramètre *URI* est inutile.

Vous pouvez également passer un préfixe de namespace personnalisé dans le paramètre *préfixe* et son URI dans le paramètre *URI*. Dans ce cas, le paramètre *URI* est obligatoire, s'il est omis une erreur est générée.

▸ Le code suivant :

SVG_ADD_NAMESPACE($svgRef;"svgNS")

... ajoute le code suivant à la racine l'objet SVG :

<xmlns="http://www.w3.org/2000/svg">

Thème : Attributs

SVG_Color_from_index

SVG_Color_from_index (index)→ Texte

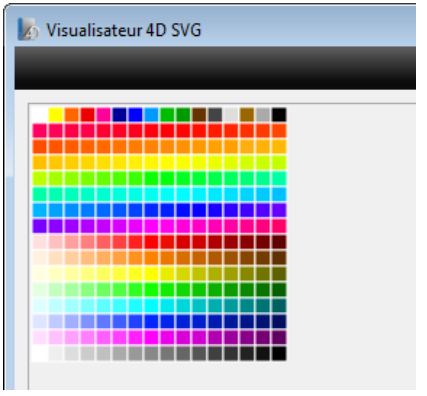
Paramètres	Type		Description
index	Entier long	→	Numéro de couleur
Résultat	Texte	←	Couleur désignée par index

La méthode SVG_Color_from_index retourne la couleur SVG correspondant à la couleur 4D définie dans le paramètre *index*.

Le paramètre *index* désigne un numéro dans la palette de couleurs de 4D, numérotée de 1 à 256.

- Dans cet exemple, on recrée la palette de couleurs de 4D :

```
$Dom_svg:=SVG_New
$Lon_line:=0
Boucle ($Lon_ii;0;15;1)
    $Lon_column:=0
      Boucle ($Lon_i;1;16;1)
      $Txt_color:=SVG_Color_from_index(($Lon_ii*16)+$Lon_i)
      $Dom_rect:=SVG_New_rect ($Dom_svg;$Lon_column;
            $Lon_line;11;11;0;0;"white";$Txt_color)
      $Lon_column:=$Lon_column+11
      Fin de boucle
    $Lon_line:=$Lon_line+11
Fin de boucle
SVGTool_SHOW_IN_VIEWER ($Dom_svg)
```



Thème : Couleurs et dégradés

SVG_Get_root_reference

SVG_Get_root_reference (objetSVG) → Ref_SVG

Paramètres	Type		Description
objetSVG	Ref_SVG	→	Référence d'objet SVG
Résultat	Ref_SVG	←	Référence de l'élément racine

La méthode SVG_Get_root_reference retourne la référence de l'élément racine de l'objet SVG dont vous avez passé la référence dans *objetSVG*.

Thème : Utilitaires

SVG_DEFINE_STYLE_WITH_ARRAYS

SVG_DEFINE_STYLE_WITH_ARRAYS (objetSVG; ptrTabNoms; ptrTabValeurs {; className {; type {; media {; titre }}}})

Paramètres	Type		Description
objetSVG	Ref_SVG	→	Référence d'objet SVG
ptrTabNoms	Pointeur	→	Pointeur vers le tableau des noms de styles
ptrTabValeurs	Pointeur	→	Pointeur vers le tableau des valeurs de styles
className	Texte	→	Nom de la classe du style CSS
type	Texte	→	Type de contenu
media	Texte	→	Descripteur de media
titre	Texte	→	Nom du style

La méthode SVG_DEFINE_STYLE_WITH_ARRAYS définit les styles de l'objet SVG désigné par le paramètre *objetSVG* à l'aide de tableaux.

- Si le paramètre *objetSVG* désigne l'élément racine, les styles sont définis en tant qu'éléments "style" inclus dans la section "defs" (*Internal Style Sheet*). Dans ce cas, le paramètre *className* est obligatoire (s'il est manquant, une erreur est retournée). Vous pouvez ensuite affecter la feuille de style *className* à des objets SVG en passant son nom à la méthode ***SVG_SET_CLASS*** (voir exemple 1).
- Si le paramètre *objetSVG* désigne un élément SVG autre que l'élément racine, le style est défini en tant qu'attribut de style pour cet élément (*Inline Style Sheet*) (voir exemple 2).

  Le paramètre optionnel *type* spécifie le langage de la feuille de style du contenu de l'élément. La valeur par défaut est "text/css".

Le paramètre optionnel *media* indique le media de destination souhaité pour l'information de style. Si vous omettez ce paramètre, la valeur par défaut utilisée est "all". Si la valeur n'est pas comprise dans la liste des types de medias reconnus par CSS2, une erreur est générée.

Le paramètre optionnel *titre* vous permet d'ajouter un attribut de type "title".

- Exemple de définition de styles internes :

```
TABLEAU TEXTE($tnoms;0)
TABLEAU TEXTE($tvaleurs;0)
AJOUTER A TABLEAU($tnoms;"fill")
AJOUTER A TABLEAU($tvaleurs;"black")
AJOUTER A TABLEAU($tnoms;"font-family")
AJOUTER A TABLEAU($tvaleurs;"'Lucida Grande' Verdana")
AJOUTER A TABLEAU($tnoms;"font-size")
AJOUTER A TABLEAU($tvaleurs;"20px")
AJOUTER A TABLEAU($tnoms;"text-align")
AJOUTER A TABLEAU($tvaleurs;"center")

$svg:=SVG_New
SVG_DEFINE_STYLE_WITH_ARRAYS($svg;->$tnoms;->$tvaleurs;"title")
$object:=SVG_New_textArea($svg;"Hello World!";10;10;200;310)
SVG_SET_CLASS ($object;"title")
```

Cette méthode génère le code suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<svg xmlns="http://www.w3.org/2000/svg">
  <defs id="4D">
    <style type="text/css">.title{fill:red;font-family:'Lucida Grande' Verdana;font-size:20px;text-align:center;}</style>
  </defs>
  <textArea class="title" height="310" width="200" x="10" y="10">Hello World!</textArea>
</svg>
```

- Exemple de définition de styles inline :

```
TABLEAU TEXTE($tnoms;0)
TABLEAU TEXTE($tvaleurs;0)
AJOUTER A TABLEAU($tnoms;"fill")
AJOUTER A TABLEAU($tvaleurs;"black")
AJOUTER A TABLEAU($tnoms;"font-family")
AJOUTER A TABLEAU($tvaleurs;"'Lucida Grande' Verdana")
AJOUTER A TABLEAU($tnoms;"font-size")
```

```
AJOUTER A TABLEAU($tvaleurs;"20px")
AJOUTER A TABLEAU($tnoms;"text-align")
AJOUTER A TABLEAU($tvaleurs;"center")

$svg:=SVG_New
$object:=SVG_New_textArea($svg;"Hello World!";10;10;200;310)
SVG_DEFINE_STYLE_WITH_ARRAYS($object;->$tnoms;->$tvaleurs)
```

Cette méthode génère le code suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<svg xmlns="http://www.w3.org/2000/svg">
  <textArea height="310" style="fill:red;font-family:'Lucida Grande' Verdana;font-size:20px;text-align:center;" width="200" x="10" y="10">Hello World!</textArea>
</svg>
```

Thème : Structures et Définitions

SVG_SET_DOCUMENT_VARIABLE

SVG_SET_DOCUMENT_VARIABLE (ptr)

Paramètres	Type		Description
ptr	Pointeur	→	Pointeur vers la variable à définir

La méthode SVG_SET_DOCUMENT_VARIABLE permet de définir un pointeur vers la variable de la base hôte qui sera mise à jour après chaque appel à SVG_SAVE_AS_PICTURE ou SVG_SAVE_AS_TEXT. Cette méthode doit être appelée une seule fois par session (par exemple dans une méthode d'initialisation).

Passez dans le paramètre *ptr* un pointeur vers la variable dont vous souhaitez suivre la valeur (généralement, la variable système *Document*).

Pour supprimer le lien, passez Nil dans le paramètre *ptr*.

Thème : Documents

SVG_SET_HUE

SVG_SET_HUE (objetSVG ; hue)

Paramètres	Type		Description
objetSVG	Ref_SVG	→	Référence d'objet SVG
hue	Entier long	→	Valeur de nuance

La méthode SVG_SET_HUE définit une valeur de nuance pour l'objet SVG désigné par le paramètre *objetSVG*. *objetSVG* doit être un conteneur SVG (svg, groupe, symbole, pattern, marqueur...) ou une image, sinon une erreur est générée.

Passez dans le paramètre *hue* une valeur située entre 0 et 360.

Thème : Couleurs et dégradés

SVG_SET_SATURATION

SVG_SET_SATURATION (objetSVG ; saturation)

Paramètres	Type		Description
objetSVG	Ref_SVG	→	Référence d'objet SVG
saturation	Entier long	→	Valeur de saturation

La méthode SVG_SET_SATURATION définit une valeur de saturation pour l'objet SVG désigné par le paramètre *objetSVG*. *objetSVG* doit être un conteneur SVG (svg, groupe, symbole, pattern, marqueur...) ou une image, sinon une erreur est générée.

Passez dans le paramètre *saturation* une valeur située entre 0 et 100.

Thème : Couleurs et dégradés

SVG_Post_comment

SVG_Post_comment (objetSVG; commentaire)→ Ref_SVG

Paramètres	Type		Description
objetSVG	Ref_SVG	→	Référence d'objet SVG
commentaire	Texte	→	Texte à ajouter en tant que commentaire
Résultat	Ref_SVG	←	Référence du commentaire

La méthode SVG_Post_comment ajoute le *commentaire* dans l'objet SVG désigné par le paramètre *objetSVG* sous forme de commentaire XML.

La méthode retourne la référence SVG du commentaire.

- Le code suivant :

C_TEXTE($comment)
$comment:="Modifié le "+**Chaine**(**Date du jour**)
$ref:= ***SVG_Post_comment*** ($svg; $comment)

... ajoute dans l'objet SVG *$svg* :

<!--Modifié le 12/12/2011-->

Thème : Utilitaires

Méthodes modifiées

SVG_SAVE_AS_PICTURE

SVG_SAVE_AS_PICTURE (objetSVG ; nomFichier {; codec})

La méthode SVG_SAVE_AS_PICTURE modifie désormais la valeur de la variable éventuellement désignée par la méthode SVG_SET_DOCUMENT_VARIABLE.

SVG_SAVE_AS_TEXT

SVG_SAVE_AS_TEXT (objetSVG *{; nomFichier}*)

La méthode SVG_SAVE_AS_TEXT modifie désormais la valeur de la variable éventuellement désignée par la méthode SVG_SET_DOCUMENT_VARIABLE.

Le second paramètre, *nomFichier*, est désormais optionnel. Omettre ce paramètre équivaut à passer une chaîne vide "" : une boîte de dialogue standard d'enregistrement de fichier apparaît, permettant à l'utilisateur de choisir le nom et l'emplacement du fichier à sauvegarder.

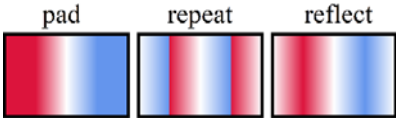
SVG_Define_linear_gradient

SVG_Define_linear_gradient (objetSVGParent ; id ; couleurDébut ; couleurFin {; rotation*{; spreadMethod{; x1; y1; x2; y2}}*})→ Chaîne

Paramètres	Type		Description
objetSVGParent	Ref_SVG	→	Référence de l'élément parent
id	Chaîne	→	Nom du dégradé
couleurDébut	Chaîne	→	Couleur de démarrage
couleurFin	Chaîne	→	Couleur de fin
rotation	Entier	→	Rotation du vecteur de dégradé
spreadMethod	*Texte*	→	*Mode de diffusion (pad, reflect ou repeat)*
x1	*Réel*	→	*Coordonnée x1 du vecteur de dégradé* *Si omis = 0*
y1	*Réel*	→	*Coordonnée y1 du vecteur de dégradé* *Si omis = 1*
x2	*Réel*	→	*Coordonnée x2 du vecteur de dégradé* *Si omis = 0*
y2	*Réel*	→	*Coordonnée y2 du vecteur de dégradé* *Si omis = 1*
Résultat	Chaîne	←	Référence du dégradé

La méthode SVG_Define_linear_gradient accepte les nouveaux paramètres optionnels suivants :

- *spreadMethod* : permet de définir le remplissage lorsque le dégradé commence ou se termine à l'intérieur des limites de l'objet *objetSVGParent*. Vous pouvez passer l'une des chaînes suivantes :
  - "pad" : utiliser les couleurs extrêmes du dégradé pour remplir le reste de la zone,
  - "reflect" : refléter indéfiniment le motif du dégradé en début/fin puis fin/début puis début/fin, etc., jusqu'à ce que l'objet soit rempli.
  - "repeat", répéter indéfiniment le motif du dégradé en début/fin, début/fin, début/fin, etc., jusqu'à ce que l'objet soit rempli.



  Si ce paramètre est omis, l'effet de la valeur "pad" est utilisé.

- *x1*, *y1*, *x2*, *y2* : ces paramètres permettent de définir le vecteur du dégradé. Ce vecteur fournit des points de départ et d'arrivée utilisés par le moteur de rendu. Vous devez passer dans ces paramètres des pourcentages exprimés sous forme de ratios (0...1).

SVG_SET_MARKER

SVG_SET_MARKER (objetSVG ; url {; position})

Vous pouvez désormais passer la chaîne "none" ou une chaîne vide dans le paramètre *url* afin de supprimer le marqueur.

SVG_CLEAR

SVG_CLEAR {(objetSVG)}

Vous pouvez désormais passer tout objet SVG valide à la méthode SVG_CLEAR (et non plus uniquement un objet SVG racine). Dans ce cas, l'objet référencé est supprimé.

SVG_New_tspan

SVG_New_tspan (objetSVGParent ; texte {; x {; y {; police | *défStyle* {; taille {; style {; alignement {; couleur}}}}}}}) → Ref_SVG

Paramètres	Type		Description
objetSVGParent	Ref_SVG	→	Référence de l'élément parent
texte	Texte	→	Texte à insérer
x	Entier long	→	Coordonnée sur l'axe x
y	Entier long	→	Coordonnée sur l'axe y
police | *défStyle*	Chaîne	→	Nom de la police *ou Définition de style*
taille	Entier	→	Taille des caractères en points
style	Entier	→	Style des caractères
alignement	Entier	→	Alignement
couleur	Chaîne	→	Couleur du texte
Résultat	Ref_SVG	←	Référence de l'objet texte SVG

Deux nouveautés concernent la méthode SVG_New_tspan :

- La méthode SVG_New_tspan peut désormais être appliquée aux éléments "textArea".
- La méthode accepte une syntaxe alternative :

  SVG_New_tspan($Dom_svg;"Hello World !"; x ; y ; style_definition)

  ... où le paramètre *style_definition* contient une définition de style complète. Si vous passez par exemple "{font-size:48px;fill:red;}", cette définition sera ajoutée en tant qu'attribut de style sous la forme :

  style="font-size:48px;fill:red;"

  Dans ce cas, les éventuels paramètres supplémentaires sont ignorés.

SVG_New_textArea

SVG_New_textArea (objetSVGParent ; texte {; x {; y {; largeur{; hauteur {; police | *défStyle* {; taille {; style {; alignement }}}}}}}) → Ref_SVG

Paramètres	Type		Description
objetSVGParent	Ref_SVG	→	Référence de l'élément parent
texte	Texte	→	Texte à insérer
x	Entier long	→	Coordonnée sur l'axe x
y	Entier long	→	Coordonnée sur l'axe y
largeur	Entier long	→	Largeur de la zone de texte
hauteur	Entier long	→	Hauteur de la zone de texte
police | *défStyle*	Chaîne	→	Nom de la police *ou Définition de style*
size	Entier	→	Taille des caractères en points
style	Entier	→	Style des caractères
alignment	Entier	→	Alignement
Résultat	Ref_SVG	←	Référence de l'objet texte SVG

La méthode SVG_New_textArea accepte désormais une syntaxe alternative :

SVG_New_textArea($Dom_svg;"Hello World !"; x ; y ; larg; haut; style_definition)

... où le paramètre *style_definition* contient une définition de style complète. Si vous passez par exemple "{font-size:48px;fill:red;}", cette définition sera ajoutée en tant qu'attribut de style sous la forme :

style="font-size:48px;fill:red;"

Dans ce cas, les éventuels paramètres supplémentaires sont ignorés.

SVG_New_text

SVG_New_text (objetSVGParent ; texte {; x {; y {; police | *défStyle* {; taille {; style {; alignement {; couleur{; rotation {; interligne{; étirement}}}}}}}}}}) → Ref_SVG

Paramètres	Type		Description
objetSVGParent	Ref_SVG	→	Référence de l'élément parent
texte	Texte	→	Texte à insérer
x	Entier long	→	Coordonnée sur l'axe x
y	Entier long	→	Coordonnée sur l'axe y
police | *défStyle*	Chaîne	→	Nom de la police *ou Définition de style*
taille	Entier	→	Taille des caractères en points
style	Entier	→	Style des caractères
alignement	Entier	→	Alignement
couleur	Chaîne	→	Couleur du texte
rotation	Réel	→	Angle de rotation du texte
interligne	Réel	→	Interlignage en point
étirement	Réel	→	Facteur d'étirement horizontal
Résultat	Ref_SVG	←	Référence de l'objet texte SVG

La méthode SVG_New_text accepte désormais une syntaxe alternative :

SVG_New_text($Dom_svg;"Hello World !"; x ; y ; style_definition)

... où le paramètre *style_definition* contient une définition de style complète. Si vous passez par exemple "{font-size:48px;fill:red;}", cette définition sera ajoutée en tant qu'attribut de style sous la forme :

style="font-size:48px;fill:red;"

Dans ce cas, les éventuels paramètres supplémentaires sont ignorés.

SVG_New_embedded_image

SVG_New_embedded_image (objetSVGParent ; image {; x {; y}}{*codec*}) → Ref_SVG

Paramètres	Type		Description
objetSVGParent	Ref_SVG	→	Référence de l'élément parent
image	Image	→	Image à inclure
x	Entier long	→	Coordonnée du coin supérieur gauche sur l'axe x
y	Entier long	→	Coordonnée du coin supérieur gauche sur l'axe y
codec	*Texte*	→	*Codec à utiliser*
Résultat	Ref_SVG	←	Référence de l'image

La méthode SVG_New_embedded_image accepte un nouveau paramètre optionnel, *codec*, permettant de définir le codec à utiliser pour l'image. Par défaut, si ce paramètre est omis, le codec est ".png".

Constantes en XLIFF

4D v13 adopte le format XLIFF pour la définition des constantes utilisées dans le code. Ces constantes étaient auparavant stockées dans des ressources de type 4DK#. Les développeurs 4D peuvent bénéficier de ce fonctionnement pour leurs propres constantes.

Notes - Le format du fichier de constantes de 4D est spécifique (basé sur des ID internes) et ne peut pas servir de base pour la création d'un fichier de constantes utilisateur. Vous devez utiliser le format décrit dans cette section pour vos propres constantes.
- Les noms des constantes personnalisées sont limités à 31 caractères.

Emplacement du fichier de constantes

Pour utiliser des constantes personnalisées dans votre base 4D, il vous suffit de :

- créer un fichier de constantes personnalisées au format XLIFF. Le nom du fichier est libre, il doit simplement comporter l'extension .xlf. Par exemple, "MesConstantes.xlf".
  Le formatage interne du fichier est décrit ci-dessous.
- placer le fichier dans le dossier **Resources** de la base ou du composant et ouvrir la base.
  Les constantes définies sont alors disponibles dans la base ouverte.

Note 4D charge également les constantes au format XLIFF placées dans le sous-dossier **Resources** du dossier **Plugins**.

Format du fichier de constantes

Imaginons que vous souhaitiez créer le thème de constantes "Mes constantes" contenant les constantes suivantes :

Constante	Type	Valeur
K_première_constante	Entier long	42
K_deuxième_constante	Texte	"this"
K_troisième_constante	Réel	3.1415

Pour cela, il vous suffit d'écrire dans le fichier XLIFF :

```
<group restype="x-4DK#">
  <trans-unit id="1">
     <source>Mes constantes</source>
  </trans-unit>
  <trans-unit id="2" d4:value="42">
     <source>K_première_constante</source>
  </trans-unit>
  <trans-unit id="3" d4:value="this">
     <source>K_deuxième_constante</source>
  </trans-unit>
  <trans-unit id="4" d4:value="3.1415">
     <source>K_troisième_constante</source>
  </trans-unit>
</group>
```

Voici la description de la syntaxe :

- restype="x-4DK#" indique que l'élément <group> définit un thème de constantes.
- Le premier </trans-unit> du groupe est utilisé comme nom de thème.
- La présence d'un attribut d4:value dans un </trans-unit> signifie qu'il s'agit d'une définition de constante. La valeur de la constante est fournie par l'attribut d4:value. Son nom est fourni par l'élément <source>.
- Le type de la constante dépend de la valeur passée dans l'attribut d4:value :
  - si la valeur ne contient que des chiffres ou le signe -, elle sera de type Entier long,
  - si la valeur est supérieure à la taille d'un Entier long, si elle contient un exposant ou une décimale, elle sera de type Réel,
  - dans tous les autres cas, elle sera de type Chaîne.

Vous pouvez cependant "forcer" le type de la constante en ajoutant ":L", ":R" ou ":S" après la valeur, par exemple : d4:value="42:S" déclare une constante Chaîne ayant pour valeur 42.

- L'ordre dans lequel les éléments </trans-unit> contenant un attribut d4:value sont placés définit l'ordre d'affichage des constantes.