4D s'ouvre sur Java

Par
Christophe KEROMEN, Cabinet de Consultants CKTI
Note technique 4D-200312-35-FR
Version 1

Date 1 Décembre 2003

Résumé

Comment accéder à 4D depuis un programme écrit en Java ?
La solution la plus immédiate passe par l'utilisation de 4D Open For Java, un produit de connectivité Java édité par 4D S.A. Nous présentons dans ce chapitre le produit lui-même, puis un exemple pour administrer à distance une application 4D. Nous tentons de décrire les principes de programmation, sans rentrer en profondeur dans de la programmation Java, ce qui dépasserait le cadre de cette note.

4D Notes techniques

4D s'ouvre sur Java

Introduction

Pour accéder à 4D depuis un programme écrit en Java deux solutions s'offrent à nous :

- à partir de la version 2003 de 4D, appeler des services web publiés par l'application 4D. SUN fournit le **Java Web Services Developer Pack**, une trousse à outils contenant toutes les classes nécessaires http://java.sun.com/webservices/ ;

- utiliser le plugin **JbyJ Pro** de Ronri-Kobo pour provoquer l'exécution distante de méthodes 4D http://www.ronri-kobo.com

Pour accéder aux données gérées par 4D, la solution la plus immédiate passe par l'utilisation de **4D Open For Java**, un produit de connectivité Java édité par 4D S.A.
Nous présentons dans ce chapitre le produit lui-même, puis un exemple pour administrer à distance une application 4D. Nous tentons de décrire les principes de programmation, sans rentrer en profondeur dans de la programmation Java, ce qui dépasserait le cadre de cette note.

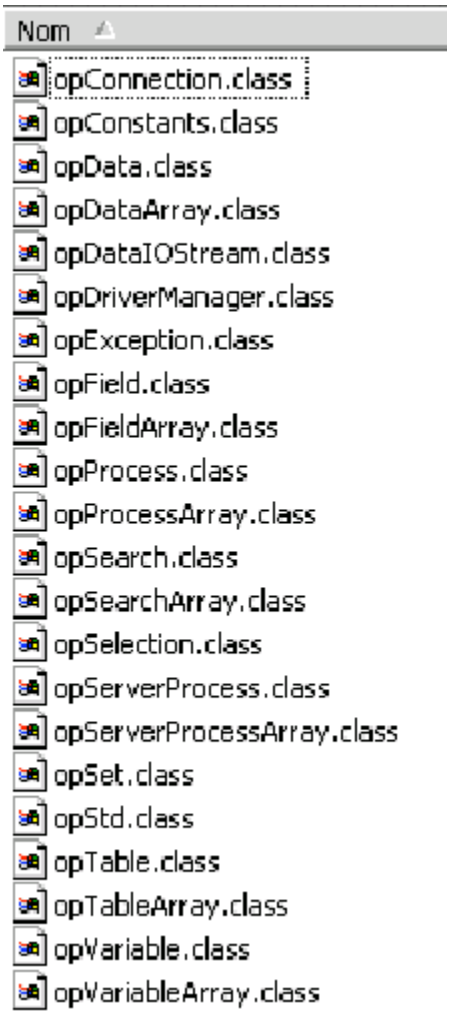
Une autre approche consiste à solliciter le serveur Web intégré à 4D en lui envoyant des requêtes HTTP. Nous ne la détaillerons pas ici car il ne s'agit que d'un cas particulier de client HTTP. Précisons simplement que la classe URLConnection de Java permet d'émettre des requêtes HTTP vers 4D Web Server. La meilleure solution est ensuite de renvoyer des données depuis 4D vers Java en les formattant en XML. Le *Java Web Services Developer Pack*, déjà cité, comprend JAXP (*Java API for XML Processing*) qui permet de traiter le flux XML reçu par le programme Java.

Note : Pour une introduction sommaire au vocabulaire et à la plate-forme Java, le développeur 4D pourra se reporter au document PDF "JAVA for 4Dummies " qui comme son nom ne l'indique pas est en français :

http://www.ckti.com/download/JavaFor4Dummies-ckti.pdf

Présentation de « 4D Open for Java »

Il s'agit d'un package Java disponible sous la forme d'une archive compressée « j4d.jar » ou d'un dossier comprenant la liste des classes distinctes. Sa faible taille (40 koctets) milite pour l'utilisation systématique de l'archive complète.



Liste des classes composant 4D Open For Java

Préfixés par "op" pour "open", la plupart des noms des classes résonnent familièrement à nos oreilles :
· Field (champ)
· Process
· Search (recherche)
· Selection
· Table
· Variable
· Array (tableau)
· Set (ensemble)
· ...
Il s'agit bien d'une traduction en Java du nom des objets manipulés quotidiennement par un développeur 4D.

4D Open : avantages... et inconvénients

4D Open For Java appartient à la famille 4D Open, cela signifie que ce produit permet d'utiliser les APIs (*Application Program Interface*) de 4D Server. Il ne pourra donc pas être utilisé pour dialoguer avec un 4ème Dimension monoposte ou un 4D Client. Il faut disposer obligatoirement d'un 4D Server.

Cette filiation a deux types de conséquences, des bonnes et des mauvaises.

Commençons par les bonnes :

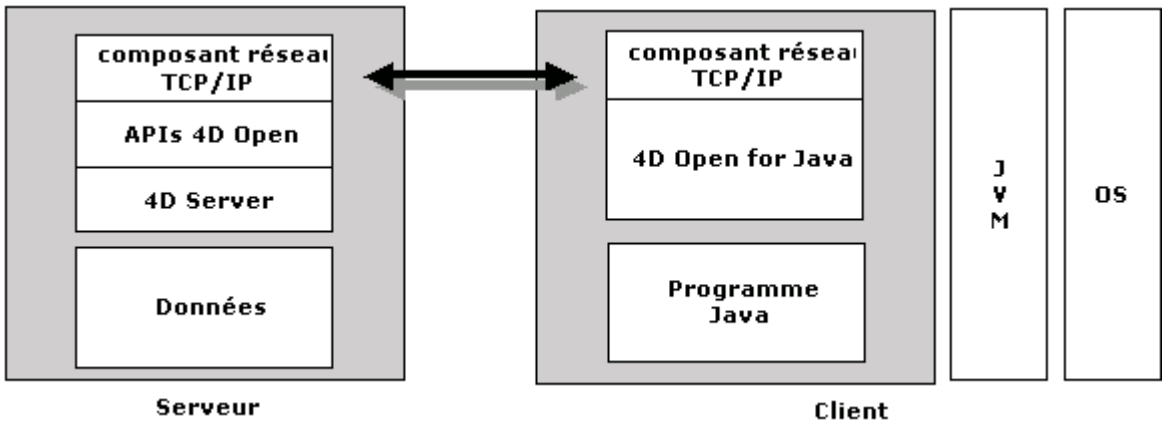
- un développeur 4D utilisant un membre de la famille 4D Open se retrouve en terrain connu. Quoiqu'en anglais, les noms de commandes sont les mêmes que celles de 4D. La logique de programmation est maintenue ;
- 4D Open est une technologie rapide, une connexion à 4D Server s'établit quasi instantanément ;
- la pérennité d'un produit édité directement par 4D S.A. paraît plus assurée.

Cependant, il faut aussi prendre en compte des aspects plus désavantageux :

- l'accès à la base de données n'est pas conforme au standard Java en la matière qui se nomme JDBC (Java Database Connectivity). Les développeurs Java doivent réapprendre une autre logique et comprendre comment fonctionne 4D, en particulier au niveau des process et des sélections. Cette lacune devrait être comblée prochainement avec la sortie prévue d'un driver JDBC pour 4D Server ;
- l'interface 4D Open suit avec du retard l'évolution de 4D Server, on n'y retrouve pas la totalité des commandes disponibles dans les dernières versions du langage 4D (compression-décompression de BLOBs par exemple) ;
- en dehors de l'exécution de procédures stockées, il est difficile de réutiliser la logique applicative développée pour la partie 4D Client, le développeur Java est amené à en dupliquer une partie de son côté.

Architecture

4D Open For Java utilisant le composant réseau de 4D Server, une couche réseau est nécessaire. Rappelons que, depuis la version 6.8, le composant réseau TCP/IP est inclus en natif dans 4D Server.



Architecture d'utilisation

Il faut également que Java soit correctement installé sur le poste client. Le poste serveur, ne recevant quant à lui que des requêtes 4D Open, n'impose aucun pré-requis concernant Java.

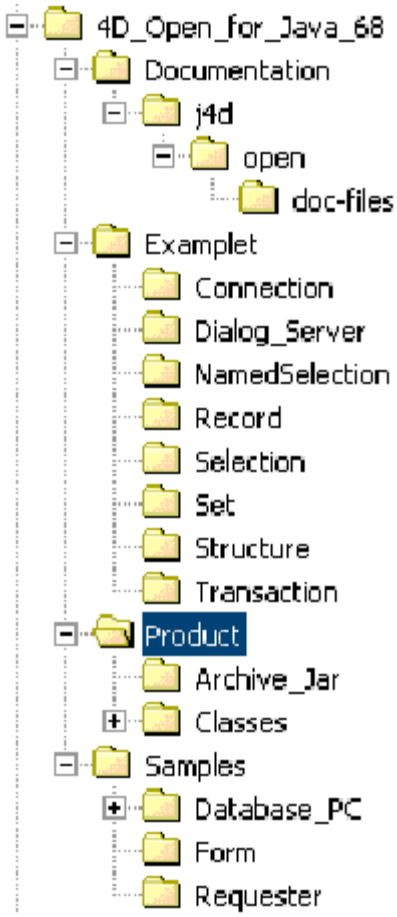
Pur Java

4D Open For Java a été certifié 100% Java par SUN. Cela valide sa totale indépendance vis à vis de la plate forme hôte entraînant sa bonne exécution aussi bien sur MacOS, Windows qu'Unix ou Linux.
Pour cela, le composant réseau a été intégré directement dans le package. Aucune installation liée à 4D n'est en conséquence requise, aspect particulièrement important dans le cas des applets.

Moyennant un peu de programmation Java, développer une application qui se connectera à un 4D Server depuis un poste tournant sous Linux se révèle possible. Nous en verrons un exemple un peu plus loin. Au contraire d'un 4D Client, cette application ne sollicitant que peu le réseau pourra tout à fait s'adapter à un fonctionnement en client distant sur des réseaux étendus ou au travers d'Internet.

Téléchargement

4D Open For Java se télécharge librement depuis le site de 4D.
Une fois l'archive téléchargée décompactée, nous obtenons le contenu suivant :



Le contenu téléchargeable sur le site de 4D

· **Documentation** contient la documentation au format HTML, nous en reparlerons au paragraphe suivant ;
· **Examplet** constitue LE dossier à explorer en priorité puisqu'il contient des exemples d'utilisation des principales méthodes de 4D Open For Java ;
· **Product** contient le produit sous forme d'une archive ou d'un répertoire de classes ;
· **Samples** comprend deux exemples et la base de données 4D cible ; à noter que ces exemples fournis uniquement en code source (fichiers .java) doivent être compilés.

Remarque : Afin de simplifier l'utilisation, il peut être efficace de rajouter le chemin d'accès vers l'archive j4d.jar dans la variable CLASSPATH. Cette variable précise l'accès aux classes les plus fréquemment accédées.

Pour plus d'informations sur la variable CLASSPATH :

java.sun.com

Licence

Une connexion 4D Open For Java consomme automatiquement une licence 4D Client. Dans cette connexion

jusqu'à six process peuvent être ouverts simultanément. Au septième process, une licence 4D client supplémentaire s'impose.

Un client 4D Open for Java peut ouvrir plusieurs connexions simultanément, soit à la même base de données, soit, contrairement à un 4D Client, à des bases de données différentes.

Documentation

La documentation publiée est au format HTML et en anglais uniquement. Vous pouvez l'obtenir en décompactant le fichier obtenu suite au téléchargement du produit. Ces pages HTML ont été générées entièrement à l'aide de **Javadoc**. Cet utilitaire Java permet de produire une documentation au format HTML, comprenant des hyperliens, en exploitant les commentaires saisis dans les programmes sources. Sous réserve de respecter un certain formalisme lors de l'écriture des commentaires, un simple appel à Javadoc rend possible la récupération d'une documentation technique dynamique.
http://java.sun.com

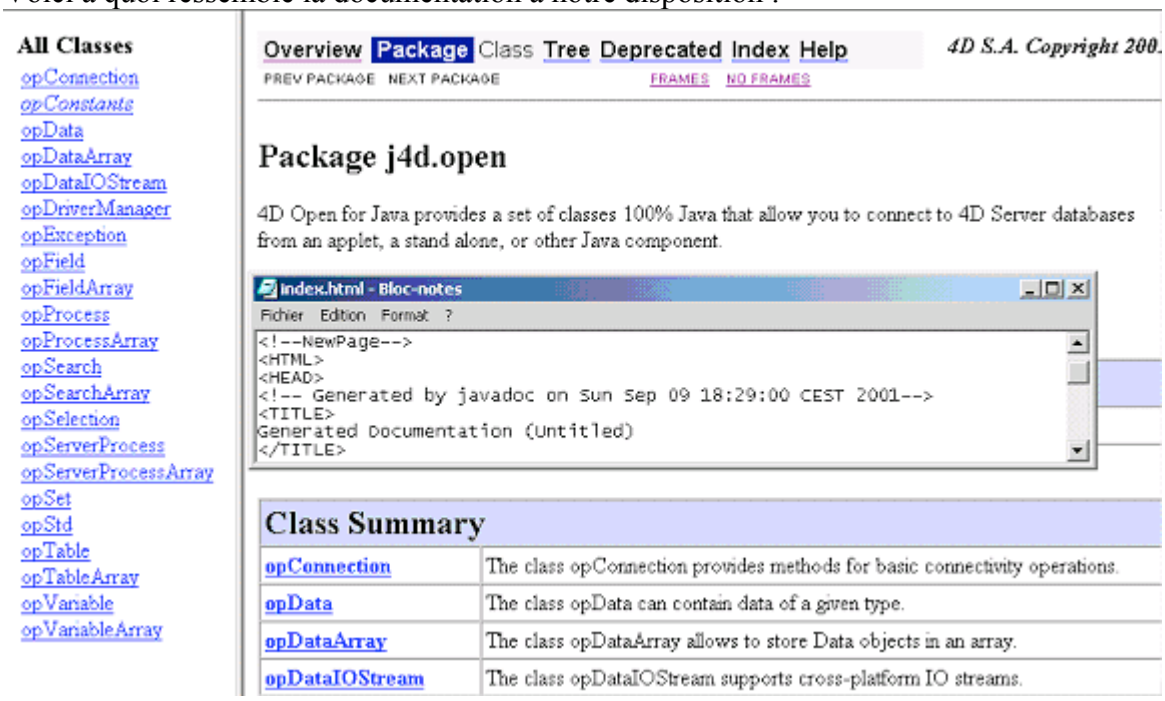
Remarque : Il est possible de générer une documentation HTML directement à partir du code des fichiers source Java sans utiliser javadoc. C'est de cette manière qu'ont été générés les pages HTML du dossier Examplet. L'utilitaire **Java2HTML** est disponible ici :

www.java2html.com

Encore plus fort, vous pouvez ensuite convertir le résultat vers une version au format des fichiers d'aide Windows grâce à **HtmlToHlp** : http://javadocs.planetmirror.com/htmltohlp.html

---

Voici à quoi ressemble la documentation à notre disposition :



Extrait de la documentation

Nous avons inséré dans la copie d'écran un extrait de l'entête HTML du fichier, montrant bien son origine.

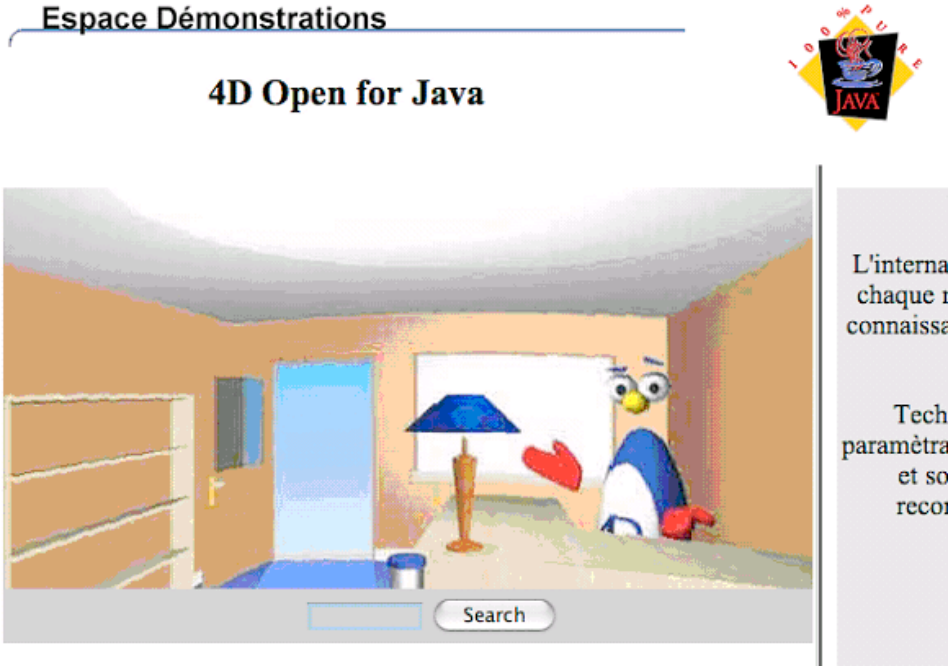
Les pages à lire en priorité :

· « help-doc.html » décrit l'organisation de la documentation ;
· « package-summary.html » propose un sommaire des classes, puis, dans la section suivante, présente la programmation avec 4D Open For Java. La lecture de cette section sera particulièrement recommandée pour les développeurs Java ne maîtrisant pas l'utilisation de 4D ;
· « overview-tree .html» présente la hiérarchie des classes.

Démonstrations

4D S.A.
Plusieurs démonstrations, sous forme d'applets, sont testables en ligne sur le site de 4D S.A., dans la section Produits/4D Open.

http://www.4d.fr/products/4dopen.html



Ces exemples montrent en particulier la taille relativement petite des applications obtenues et la rapidité des connexions 4D Open For Java (à distinguer de la relative lenteur du téléchargement initial de l'applet Java). Comme nous l'avons déjà, rien de comparable avec le temps de connexion d'un 4D Client à 4D Server.
En pratique, il est tout à fait envisageable de fonctionner en mode déconnecté, l'application Java ne se connectant que ponctuellement à 4D Server pour y lire ou écrire des informations.
Cela présente un net avantage au niveau du nombre de process tournant sur le serveur et du coût des licences à acquérir pour permettre un nombre important de connexions potentielles.

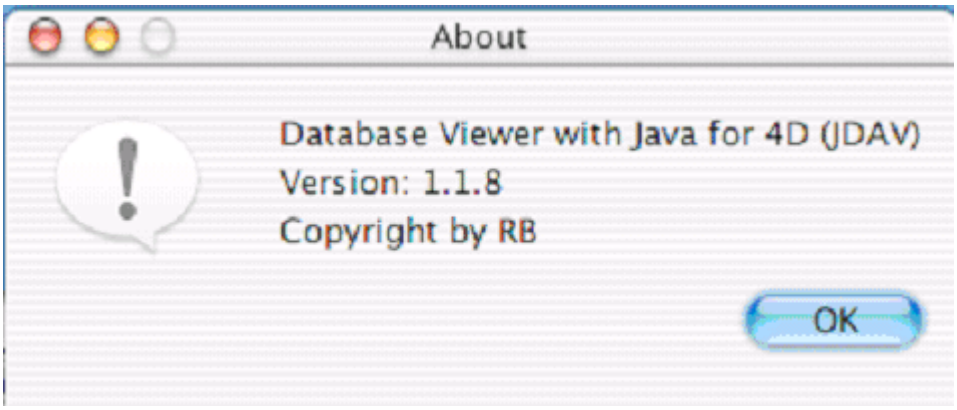
PLANÈTE 4D : WRITEREADER

Dans le numéro 7 de cette revue, nous avons conçu, avec Pascal Pradier de chez 4D S.A., un petit programme Java permettant :
· d'interroger un 4D Server pour lui demander une liste des documents 4D Write gérés ;
· de sélectionner un document dans cette liste ;
· de convertir sur le serveur ce document au format Word ;
· de ramener ce document converti sur le poste client ;
· de lancer MS Word pour consulter le document.

Le client Java ne se connecte au serveur qu'à deux moments :
· pour obtenir la liste des documents 4D Write présents sur le serveur ;
· pour demander à 4D d'exécuter la procédure stockée de conversion de 4D Write en RTF.

Le point important de cet exemple réside dans la répartition des tâches entre client et serveur et donc entre Java et 4D. Il est plus rapide d'écrire du code 4D que du code Java. Il y a donc tout intérêt à demander, depuis le programme Java, l'exécution de procédures stockées écrites en 4D sur le serveur 4D, puis à récupérer le résultat.

JAVA DATABASE VIEWER FOR 4D
Voici la description d'un projet encore en cours de développement, mais qui illustre fort bien les capacités de 4D Open For Java .

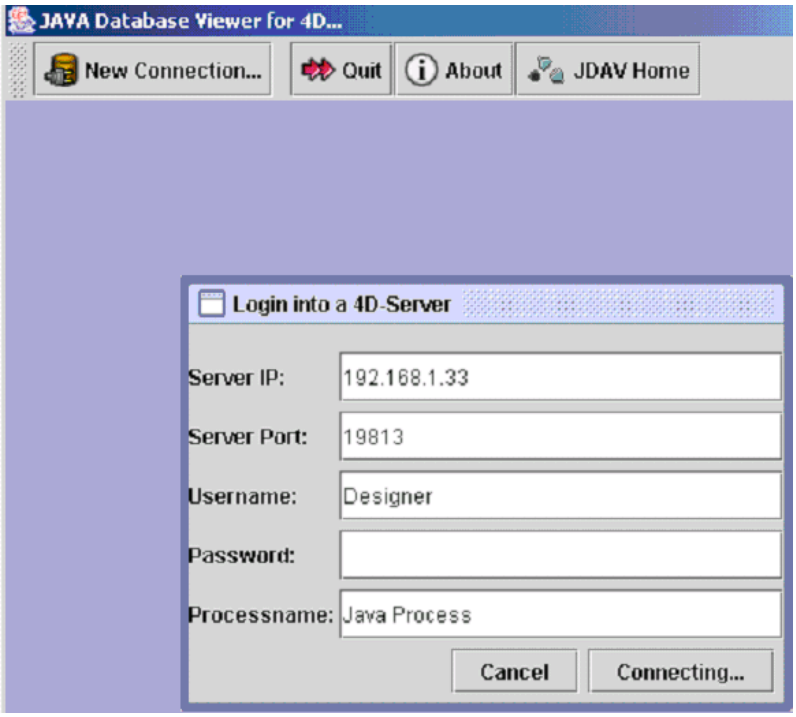


Java Database viewer for 4D

Cette petite archive Java de 100 k constitue un client universel à toute base servie par un 4D Server ! Vous pouvez la télécharger ici :
http://www.hitech-4d.com

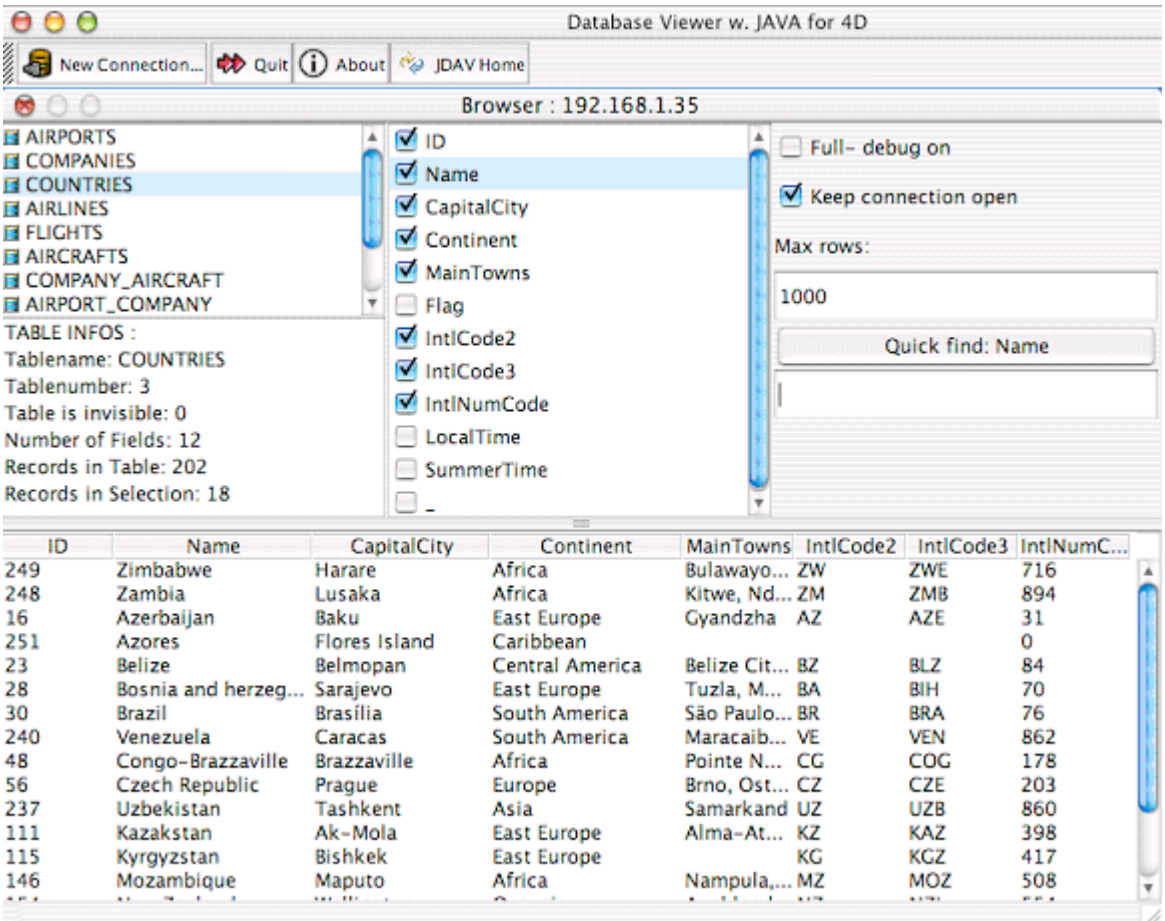
Nous avons testé la version 1.1.8.
Le développeur a également retenu un fonctionnement par défaut en mode déconnecté. L'application ne se connecte au 4D Server que le temps d'effectuer sa requête, de recevoir le résultat puis se déconnecte aussitôt. Une option permet toutefois de demander à conserver la connexion le temps de la session afin d'optimiser les temps de réponse.

En choisissant « New Connection… », un premier écran permet de se connecter à un 4D Server en spécifiant les différents paramètres de la connexion.



écran de connexion

L'application se connecte alors à 4D Server et présente la liste des tables dans un browser.
Un clic sur une des tables affiche, après un petit temps de latence (connexion au serveur puis rapatriement des enregistrements et construction de la grille), la liste des enregistrements.
Nous sommes ici connectés à la base 4D Airport dont nous visualisons les données de la table « COUNTRIES».

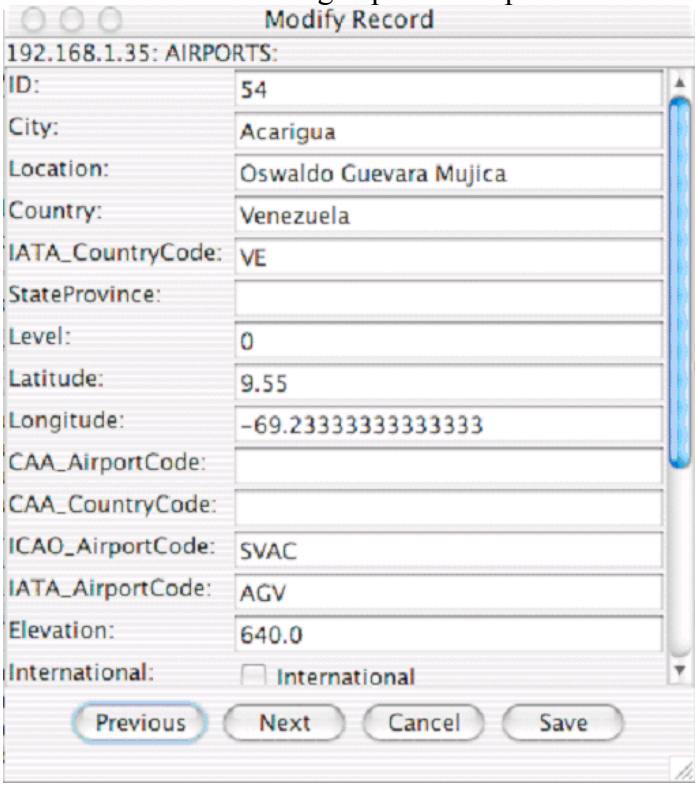


Liste des enregistrements

Notez, en haut à gauche, les informations sur la table sélectionnée. Sur la droite de l'écran, le bouton « Quick find » permet d'effectuer une recherche mono-critère.

Les enregistrements sont présentés en utilisant un contrôle *TableView* de type grille, propre à Java et très fonctionnel (colonnes redimensionnables, tris, saisie,…).

Un double-clic sur une ligne permet de passer en saisie.



Modification d'un enregistrement

Tous les écrans sont, bien entendu, construits dynamiquement en fonction des informations lues sur la table (liste des champs, nombre d'enregistrements,…). La conversion des différents types de données s'effectue automatiquement. La base n'a subi aucune modification pour être requêtable par ce client. Mieux, il s'agit d'une base servie par un 4D Server 4D 2003, alors que cette version de JDAV est antérieure à la sortie de l'opus 2003. En fait, JDAV permet de se connecter à toute version de 4D Server depuis la 6.7.

Il s'agit d'un projet encore au stade de développement et cependant très prometteur.
Il lui manque en particulier des possibilités étendues de recherche-tri (prévues dans une prochaine version 1.2) pour obtenir un équivalent parfait au mode utilisation directe complètement multi plateforme et qui tienne dans environ 100 k !
On pourrait peut-être souhaiter qu'un tel projet, dont toute la communauté 4D bénéficierait, puisse profiter d'une dynamique de développement de type open-source.

Programmation avec 4D Open For java

Principes de base

La classe **opProcess** constitue le socle de la programmation avec 4D Open For Java. Y sont listées, un peu en vrac, les principales méthodes de gestion :

- des ensembles ;

- des sélections ;
- des transactions ;
- des sémaphores ;
- des enregistrements ;
- d'exécution de procédures stockées ;
- des process ;
- des liens ;
- des recherches.

La communication avec 4D Server pourra s'effectuer au moyen :
- des enregistrements de la base de données ;
- de l'exécution de procédures stockées ;
- d'une communication interprocess par la lecture/écriture du contenu des variables de 4D Server.

En Java, toute méthode appartient à une classe, soit sous forme de méthode statique, soit comme méthode d'instance.

Les méthodes de 4D Open For Java sont principalement des méthodes d'instance et nécessitent en conséquence l'instanciation d'un objet préalablement à leur exécution.
Contrairement à 4D, aucun process n'est ouvert par défaut et aucune méthode ne peut être utilisée sans instanciation de son objet de rattachement.

Exemple de Session
Ci-dessous un code très simplifié présentant les principales étapes d'une session de travail.

Connexion

OpConnection connect=null ;

Initialisation d'un objet **opDriverManager** nécessaire pour pouvoir appeler la méthode **getconnection** effectuant la connexion à 4D Server, sous réserve de lui passer une adresse IP et un numéro de port.

```
OpDriverManager drvManager = new opDriverManager() ;
connect = drvManager.getconnection (vIpAdr,vNumPort) ;
```

Création d'un process

```
opProcess p = null ;
```

Ici encore nous instancions un objet pour appeler une de ses méthodes, en l'occurrence pour démarrer un nouveau process, contexte de base de toute opération 4D Open For Java. Notez que l'identification s'effectue au moment de démarrer le process et non lors de la connexion.

```
p = connect.startProcess (connectionName, userName, password, processname) ;
```

Sélections, mises à jour, etc.

Voir le paragraphe suivant pour un exemple détaillé de recherche.

Fermeture du process

```
connect.stopProcess(p) ;
```

Vous avez bien vu, le process demande lui-même à se détruire ! Une forme de suicide que n'autorise pas 4D Client.

Déconnexion

connect.CloseConnection() ;

Attention, la méthode **getConnection** appartient à un objet **opDriverManager** tandis que la méthode **CloseConnection** (attention à la capitale) s'obtient à partir de la classe **opConnection**.

Cet exemple minimal est mis en pratique dans la classe **Test1** de l'exemple doConnect. Voici le suivi de son exécution sur la console :

Demande d'exécution de l'exemple en fixant les différents chemins d'accès :

```
C:\JAVAAPP\TEST_J4D\DOCONNECT>set SOURCE_DIR=C:\JavaApp\Test_j4d\doConnect\

C:\JAVAAPP\TEST_J4D\DOCONNECT>set CLASSPATH=.;C:\JavaApp\Test_j4d
\doConnect\;C:\JavaApp\Test_j4d\j4d.jar

C:\JAVAAPP\TEST_J4D\DOCONNECT>java -cp .;C:\JavaApp\Test_j4d
\doConnect\;C:\JavaApp\Test_j4d\j4d.jar Test1 "192.168.1.33"
```

Notez le passage de l'adresse IP du 4D Server comme argument à la classe **Test1**.

La classe affiche un message sur la console indiquant que la connexion s'est bien déroulée :

```
connexion et process ouvert sur 192.168.1.33
```

Recherche

Présentation de la recherche

Pour illustrer la section « Sélections, mises à jour, etc. » de l'exemple de session précédent, nous allons procéder à une recherche sur un 4D Server publiant la base de données 4D Airports.
Nous souhaitons connaître le nom de l'aéroport connaissant sa ville.

La recherche portant sur la table AIRPORTS s'écrirait en 4D :

Exemple de recherche en 4D

```
$Criteria:="Abidjan"

CHERCHER ([AIRPORTS];[AIRPORTS]City=$Criteria)
ALERTE("Le nom de l'aéroport de '"+$Criteria+"' est '"+[AIRPORTS]Location+"'")
```

Le code Java

Nous allons décrire son équivalent en Java, en supposant que nous sommes déjà connecté à 4D Server et qu'un process vProcess a été créé.

Exemple de recherche en Java

Nous commençons par déclarer un objet à une dimension **mySearch** de type **opSearchArray**. La dimension correspond au nombre de lignes de la requête.

```
opSearchArray mySearch=new opSearchArray(1);
```

Chaque ligne d'un objet **opSearchAray** correspond à un objet décrivant un critère de recherche. Nous le remplissons en lui passant dans l'ordre : opérateur, n° table, n° champ, opérateur de recherche, data à chercher.

```
mySearch.mSearchArray[0]=new opSearch(NONE,1,2,EQUAL,new opData (ALPHANUMERIC,myCity));
```

La requête se déclenche en appelant la fonction **Search** de l'objet **vProcess** et en lui passant l'objet **mySearch** qui décrit la requête à effectuer.
Le résultat **vresult1** est un objet de type **opSelection**.

```
opSelection result1=vProcess.Search(mySearch);
```

Nous allons afficher sur la console la propriété cardinale de cette sélection **mRecordsInSelection**.

```
nbElemSel=result1.mRecordsInSelection;
System.out.println("Nbre d'aéroport trouvé(s) pour "+myCity+" :"+nbElemSel);
```

Nous souhaitons maintenant lire le troisième champ ([AIRPORTS]Location) de l'enregistrement courant de la table.
Pour cela nous devons décrire ce champ dans un objet à une dimension (car un seul champ souhaité) de type **opFieldArray**.

```
opFieldArray vFieldArray=new opFieldArray(1);
```

Remplissons l'objet décrivant le champ en lui indiquant le n° de la table et du champ cible.

```
vFieldArray.mTargetTable=1;
vFieldArray.mFieldArray[0]=new opField(1,3);
```

Notons que nous devons transformer pour cela le champ 3 de la table 1 en un objet de type **opField**.

Nous définissons maintenant un objet à une dimension (toujours pour un seul champ) de type **opDataArray** qui sera le tableau recevant les données du champ.

```
opDataArray ArrayData=new opDataArray(1);
```

Chargeons les données du champ dans notre tableau **ArrayData** grâce à la méthode **LoadFields** de l'objet **opProcess** en lui passant en paramètre l'objet décrivant le champ.

```
ArrayData=vProcess.LoadFields(vFieldArray);
```

Il ne nous reste plus qu'à afficher sur la console la valeur relue, de type String (chaîne de caractère), dans l'élément 1 (à la position 0) de l'objet.

```
System.out.println ("Le nom de l'aéroport de '"+myCity+"' est : '"+
ArrayData.mDataArray[0].mString+"'");
```

Exécution de l'exemple

Demander l'exécution de l'exemple en fixant les différents chemins d'accès :

```
C:\JAVAAPP\TEST_J4D\DOSEARCH>set SOURCE_DIR=C:\JavaApp\Test_j4d\doSearch\

C:\JAVAAPP\TEST_J4D\DOSEARCH>set CLASSPATH=.;C:\JavaApp\Test_j4d
\doSearch\;C:\JavaApp\Test_j4d\j4d.jar
```

Puis en appelant le nom de la classe suivi des différents paramètres séparés par des espaces :

```
C:\JAVAAPP\TEST_J4D\DOSEARCH>java -cp.;C:\JavaApp\Test_j4d\doSearch\;
C:\JavaApp\Test_j4d\j4d.jar doSearch "192.168.1.33" "Abidjan"
```

Compte-rendu d'exécution :

```
connexion et process ouvert sur 192.168.1.33
Nbre d'aéroport(s) trouvé(s) pour Abidjan :1
```

Conseils pratiques

Quoiqu'écrire de petits programmes en utilisant simplement un éditeur de texte soit possible, dès que l'on aborde les classes graphiques il est plus efficace d'utiliser un IDE spécialisé, de type JBuilder (Ed. Borland : http://www.borland.fr/jbuilder/)

Même si le code est réellement multi-plateforme, des tests approfondis sur les différents OS permettront de détecter les problèmes liés aux différences de version entre les JVM.

Bien séparer, lors de la conception, les classes d'accès aux données de celles gérant l'interface graphique. Outre la facilité de mise au point de la partie concernant 4D Open For Java, cela vous permettra éventuellement de déléguer l'habillage à un programmeur Java plus expérimenté, tout en lui évitant de trop se plonger dans 4ème Dimension.

En fait, tirer parti de ce produit sera plus facile pour un développeur Java ne connaissant pas 4D, que pour un développeur 4D qui devrait apprendre le langage Java. Notre conseil consistera à préférer former un développeur Java expérimenté au fonctionnement du moteur de 4D, ce qui lui permettra ensuite une prise en main de 4D Open For Java assez rapide.

Nous avons déjà conseillé, dans la partie concernant les démonstrations, d'équilibrer la collaboration Java/4D, de manière à tirer au maximum parti de l'exécution de procédures stockées sur 4D Server.