

Date	25-3-2003
Auteur	CK
rév	4 28/8/2004

4D FAIT LA JAVA

4D fait la java	1
Introduction	1
JExternal	2
Transtypage 4D-Java	3
Architecture des fichiers	3
Initialisation de la JVM	3
Appel d'une méthode statique	4
méthode statique et méthode d'objet	4
Si ce n'est pas statique...	5
2+2= ?	5
Appel d'une méthode d'objet	7
Pseudo-code d'appel	7
Création de l'objet	9
Appel d'une méthode call-back	10
Exécution de javascript	11
Conclusion	12
Autres démos fournies par Ronri Kobo	13
JEX_JDBCDemo	13
JEX_FTPDemo	13
Création de PDF depuis un document XML	13
Autres plug-ins de Ronri-Kobo	17

INTRODUCTION

Pourquoi se préoccuper de Java quand on programme en 4D ?

Apportons une première réponse en précisant que Java est généralement LE langage objet enseigné dans les cursus de formation à l'informatique. Notre métier nous conduisant à côtoyer des professionnels d'univers différents, il est fructueux de s'intéresser aux autres langages utilisés. Une connaissance même superficielle de la Programmation Orientée Objet (POO) représente un atout de moins en moins négligeable. Si vous êtes amenés à embaucher un jeune diplômé, le dialogue sera plus facile à établir. Même s'il s'agit sournoisement de l'inciter par la suite à programmer en 4D !

En outre, pour un développeur formé à Java, la migration vers le nouveau langage C# de Microsoft s'effectue aisément, ouvrant la porte du *framework* .NET, sésame de la programmation moderne sous Windows. Enfin, Goldfinger, le nouvel outil de développement de 4D S.A., disposera d'un langage orienté objet.

Une deuxième motivation, plus pragmatique, peut provenir du constat que les groupes de développement *open-source* utilisent énormément Java. Il est tentant de réutiliser une partie de leurs travaux dans le cadre d'un projet 4D. C'est le cas dans le domaine XML, où en dépit des nouveautés intégrées à 4D 2003, des lacunes importantes subsistent, comme l'absence de processeur XSL-T ou FOP. Or, d'excellents programmes JAVA de ce type sont disponibles : nous verrons à la fin de ce document qu'il est tout à fait possible de les appeler à partir de 4D.

La nécessité d'intégrer une solution fondée sur 4D dans un système utilisant des composants développés en Java peut également constituer une raison valable.

Pour une introduction au vocabulaire Java, se reporter au document « Java for 4Dummies » du même auteur : <http://ckti.com/download/JavaFor4Dummies-ckti.pdf>

Distinguons trois types de collaboration 4D/Java :

- un programme Java souhaite accéder aux données 4D : la solution passera par deux produits de 4D S.A. : 4D Open For Java ou le nouveau driver JDBC;
- un programme Java souhaite appeler des méthodes 4D : la solution sera le plug-in **Jbyl Pro** de la société Ronri-Kobo ;
- un programme 4D souhaite appeler des méthodes Java : la solution sera le plug-in **JExternal** de la même société .

Ce document se consacre à la description de ce dernier type de connectivité.

JDBC est un standard permettant aux programmes Java la connexion à des bases SQL, indépendamment du moteur cible. C'est l'équivalent dans le monde Java d'ODBC.

JEXTERNAL

La seule solution à ce jour pour appeler des classes Java à partir de 4D consiste à utiliser le plug-in **JExternal** édité par la société japonaise Ronri Kobo : www.ronri-kobo.com.

Java+plug-in+Japon : ne vous laissez pas effrayer ! Ajoutons que cet outil permet également d'exécuter du code Javascript depuis 4D, et même de se connecter à une base Oracle via un driver JDBC, comme Bruno Legay l'a démontré lors d'une conférence à l'université 4D de 2001. Votre curiosité se ravive, non ?

JExternal, disponible en version windows et macOS carbonisée, fonctionne en mode démonstration pendant une heure. Le support de l'éditeur (en anglais et en japonais) est efficace et réactif.

La documentation fournie avec le plug-in (en anglais et en japonais) est plutôt sommaire et la base exemple d'appels de méthodes Java depuis 4D ne facilite guère la prise en main. Nous allons détailler la manière d'appeler une méthode Java depuis 4D, de passer des paramètres, de recevoir les valeurs retournées et d'effectuer le rétro-appel d'une méthode 4D (*callback*).

La seule limitation d'une méthode Java appelée par 4D consiste à ne pas utiliser d'objets d'interface. Autrement dit, 4D gère l'interface tandis que les méthodes Java sont utilisées comme routines de manière transparente pour l'utilisateur.

Si une méthode Java doit interagir avec ce dernier, elle effectuera un rétro-appel (*callback*) à une méthode 4D, (voir l'exemple page <ref>#</ref>).

Comme il demeure plus productif de réaliser des interfaces graphiques avec 4D plutôt qu'en Java, cette collaboration s'avère efficace.

TRANSTYPAGE 4D-JAVA

Lors des échanges de données, JExternal assure automatiquement le transtypage entre les types de 4D et ceux de Java.

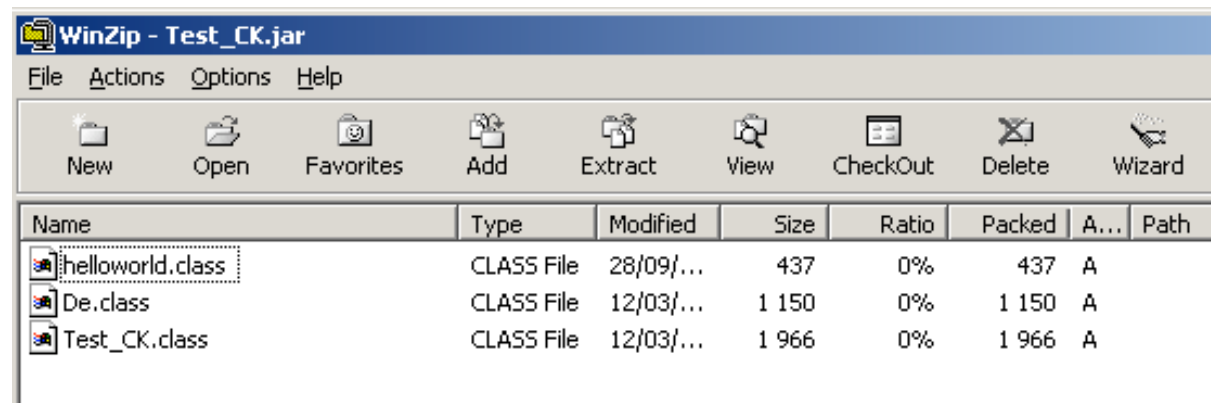
Pour se rapprocher des types 4D, ronri-kobo a rajouté aux types de base Java la gestion :

- des heures : com.ronri_kobo.Time ;
- des dates : com.ronri_kobo.Date ;
- des alphas : com.ronri_kobo.String255;
- des images : com.ronri_kobo.Picture.

ARCHITECTURE DES FICHIERS

La structuration des différents fichiers nécessaires est précise, au même niveau que la structure de la base doivent se trouver trois dossiers :

- RonriKobo : contient les packages utilitaires utilisés en interne par JExternal ;
- Mac4DX ou Win4DX : contient deux plug-ins, JExternal et JWorld ;
- JavaClasses : contient les packages comprenant les classes qui seront appelées par JExternal, par exemple « Test_CK.jar ».



Exemple de `.jar<file>Jar.tif</file>`

Attention.

JExternal ouvre en écriture les packages contenus dans le dossier JavaClasses. Pour effectuer une modification de leur contenu (remplacer une ancienne classe par sa nouvelle version) vous devrez quitter 4D, ce qui n'est pas très pratique.

INITIALISATION DE LA JVM

Ne passons pas sous silence le travail à effectuer lors de l'initialisation de l'application, il faut :

- sur Windows, indiquer le chemin d'accès à la JVM à utiliser, tandis que sur MacOS « JVM » suffit ;
- avec 4D Client, recopier en local les dossiers « RonriKobo » et « JavaClasses », l'éditeur fournissant les routines 4D nécessaires dans la base de démonstration ;
- initialiser une machine virtuelle par l'appel suivant :
`$_e_JVM_ID := JWD Create java World ($_s_JVM_Path)`
 où la variable texte `$_s_JVM_Path` contient le chemin d'accès à la machine virtuelle ;
- initialiser l'environnement (mode démonstration ou non) en renseignant le n° de licence d'utilisation :
`$_E_Error := JEX Set environment ($_e_JVM_ID ; $_s_JEX_Licence)`
 où la variable texte `$_s_JEX_Licence` contient le n° de licence du plug-in.

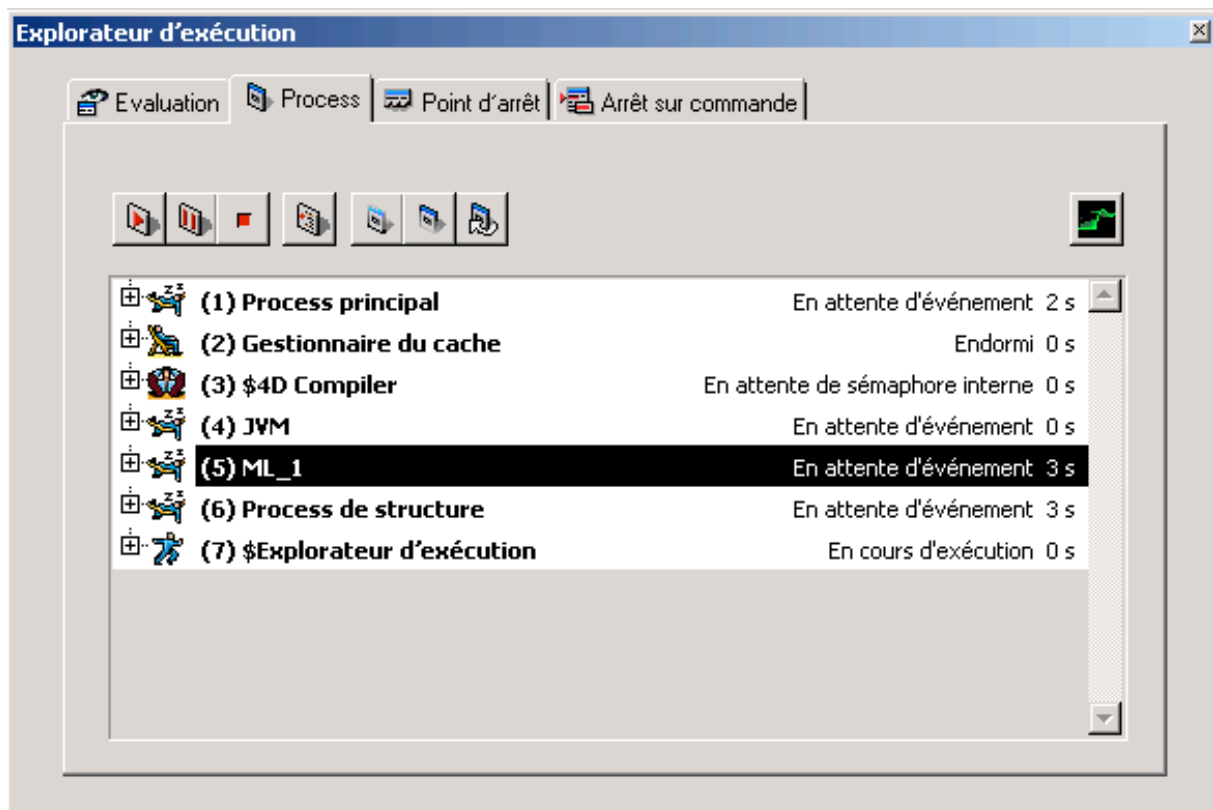
La méthode base « Sur Ouverture » constitue l'emplacement idéal pour ces appels préalables.

Remarque.

Il suffit d'initialiser une machine virtuelle pour toute la session.

Pour éviter des erreurs fatales sur 4D Server, libérez la machine virtuelle avant de quitter en appelant la commande **JWD Close java world** dans la méthode base « Sur fermeture serveur ».

L'appel à **JWD Create java World** a pour conséquence la création d'un process supplémentaire nommé JVM :



Process JVM<file>ProcessJVM.tif</file>

APPEL D'UNE MÉTHODE STATIQUE

MÉTHODE STATIQUE ET MÉTHODE D'OBJET

Sans grand rapport avec les méthodes objet du monde 4D, Java distingue la **méthode d'objet**, dépendante d'un objet (instance d'une classe), de la **méthode statique**, propre à la classe elle-même.

Une méthode retournant le nombre d'objets de la classe est par exemple de type statique, à la différence d'une méthode retournant la valeur d'un attribut d'un objet.

SI CE N'EST PAS STATIQUE...

L'exécution d'une méthode d'objet pourra donc dépendre de l'état de l'objet auquel elle appartient, caractérisé par la valeur courante de ses attributs ou propriétés, appelés également variables d'instances.

Autrement dit, une méthode liée à l'objet 1 pourra se comporter différemment de la même méthode liée à l'objet 2, bien qu'objet 1 et objet 2 appartiennent à la même classe et que le code de la méthode soit identique.

2+2= ?

Le défi consiste à appeler une méthode Java qui effectuera la somme des deux entiers passés en paramètres !

Outre le fait de vérifier si Java sait calculer correctement, cela nous permettra de :

- valider le transtypage effectué automatiquement par JExternal entre 4D et Java;
- fournir le squelette d'une méthode d'appel.

Voici le code Java de la classe « Test_Addition » : la méthode statique qui nous intéresse se nomme « add_LongIntVar_c ».

Code source de la méthode statique

```
/******  
// Test_CK Class  
/******  
public class Test_CK extends Object  
{  
/******  
    // add_LongIntVar_c : ex. d'appel statique  
    //additionne deux entiers et retourne le résultat  
    /******  
    public static int add_LongIntVar_c( int param1, int param2 )  
    {  
        int result = param1+param2;  
        return result;  
    }  
}
```

Ci-dessous le pseudo-code correspondant à la méthode d'appel 4D :

- Déclaration et remplissage d'un tableau de pointeurs avec les deux entiers à additionner ;
- Appel de la routine de JExternal **JEX Call class method** qui exécute une méthode statique de la classe spécifiée;
- Affichage du résultat transmis par ré-utilisation du tableau de pointeurs.

La méthode **JEX Call class method** attend trois paramètres :

- nom de la classe : "Test_CK" ;
- nom de la méthode statique à exécuter : "add_LongIntVar_c" ;
- tableau de pointeurs vers les paramètres à passer à la méthode : ici les deux entiers à additionner.

Elle retourne un entier différent de zéro en cas d'erreur ainsi que des valeurs, par l'intermédiaire du tableau de pointeurs passé en troisième paramètre.

Pas de quoi s'affoler !

Code d'appel 4D

`Appel d'une méthode statique de classe

`Ajout de deux entiers en appelant Test_CK.add_LongIntVar_c()

C_TEXTE(\$1)

C_TEXTE(\$_s_ClassName;\$_s_MethodName)

C_ENTIER(\$_e_Error;\$_e_Result)

C_TEXTE(\$_s_entier1;\$_s_entier2)

C_ENTIER LONG(ck_e_param1;ck_e_param2)

ck_e_param1:=2 `Interne à cette méthode

ck_e_param2:=2

\$_s_ClassName:="Test_CK"

Si (Nombre de parametres=0)

\$_s_MethodName:="add_LongIntVar_c"

Sinon

\$_s_MethodName:=\$1

Fin de si

Repetier

`préparation du tableau de passage de paramètres

TABLEAU POINTEUR(\$_Tp_Parameters;2)

\$_Tp_Parameters{1}:=>ck_e_param1

\$_Tp_Parameters{2}:=>ck_e_param2

`Appel de la méthode statique

\$_e_Error:=**JEX Call class method** (\$_s_ClassName;\$_s_MethodName;\$_Tp_Parameters)

`Traitement du résultat

Si (\$_e_Error=0)

Si (Taille tableau(\$_Tp_Parameters)=1)

Si (Type(\$_Tp_Parameters{1}->)=Est_un_entier_long_)

\$_e_Result:=\$_Tp_Parameters{1}->

Si (Nombre de parametres=0)

ALERTE(Chaine(ck_e_param1)+" + "+Chaine(ck_e_param2)+" = "+Chaine(\$_e_Result))

Sinon

`le résultat est affiché par la méthode Java via un callback à wrapper_Alerte

Fin de si

Fin de si

Tél.: 02 97 40 13 27

Site web : www.ckti.com

Contact : ckti@ckti.com

```

    Fin de si
Sinon
    ALERTE(JEX_ErrMsg ($_e_Error))
Fin de si

CONFIRMER("Une autre addition d'entiers ?")
Si (ok=1)
    $_s_entier1:=Demander("Entier 1";Chaine(ck_e_param1))
Fin de si
Si (ok=1)
    $_s_entier2:=Demander("Entier 2";Chaine(ck_e_param2))
Fin de si

Si (ok=1)
    ck_e_param1:=Num($_s_entier1)
    ck_e_param2:=Num($_s_entier2)
Fin de si
Jusque (ok=0)

ck_e_param1:=0
ck_e_param2:=0

```

APPEL D'UNE MÉTHODE D'OBJET

PSEUDO-CODE D'APPEL

4D n'étant pas un langage objet, l'appel à une méthode d'objet nécessite un travail supplémentaire pour créer et identifier l'objet concerné. Nous devons obtenir et utiliser un identifiant de l'objet instancié. Cet identifiant, sous forme d'entier long, référencera l'objet de manière unique pour toute manipulation ultérieure.

Voici le pseudo-code de la méthode 4D :

Création d'une instance de l'objet et récupération de son identifiant

- Déclaration et remplissage d'un tableau de pointeurs ;
- Appel de la routine **JEX Call class method** pour exécuter la méthode statique de la classe cible qui crée l'objet ;

Exécution de la méthode d'objet

- Remplissage d'un tableau de pointeurs de paramètres ;
- Appel de la routine **JEX Call object method** pour exécuter une méthode d'objet de l'objet référencé ;
- Affichage du résultat transmis par ré-utilisation du tableau de pointeurs ;
- Libération de l'objet créé, par appel de la routine **JEX Free object** à laquelle on passe l'identifiant de l'objet à libérer.

Exemple.

Dans la base exemple, ce schéma est utilisé pour appeler les méthodes lancer() et getFacevalue() d'une classe **De**, retournant ainsi une valeur aléatoire entre 1 et 6.

Code de la méthode 4D d'appel

```

`Ajoutte deux entiers en appelant Test_CK.add_LongIntVar_o()

C_ENTIER LONG($_e_Result;$_e_TargetObjetID;$_e_Error)
C_TEXTE($_s_ClassName;$_s_MethodName)
C_TEXTE($_s_entier1;$_s_entier2)

C_ENTIER LONG(ck_e_param1;ck_e_param2)
ck_e_param1:=2 `variables internes à cette méthode

ck_e_param2:=2

$_s_ClassName:="Test_CK"
$_s_MethodName:="add_LongIntVar_o"

`____ Création de l'objet et récupération de son ID

$_e_TargetObjetID:=JEX_CreateObject ($_s_ClassName;"createMyself")

Si ($_e_TargetObjetID#0)
  Repeter
    `préparation du tableau de passage de paramètres

    TABLEAU POINTEUR($_Tp_Parameters;2)
    $_Tp_Parameters{1}:=>ck_e_param1
    $_Tp_Parameters{2}:=>ck_e_param2

    `Appel de la méthode d'objet

    $_e_Error:=JEX_Call object method ($_e_TargetObjetID;$_s_MethodName;$_Tp_Parameters)

    `Traitement du résultat

    Si ($_e_Error=0)
      Si (Taille tableau($_Tp_Parameters)=1)
        Si (Type($_Tp_Parameters{1}->)=Est_un_entier_long)
          $_e_Result:=$_Tp_Parameters{1}->
          `le résultat est affiché par la méthode Java via un callback à wrapper_Alerte
        Fin de si
      Fin de si
    Sinon
      ALERTE(JEX_ErrMsg ($_e_Error))
    Fin de si
  
```

```
CONFIRMER("Une autre addition d'entiers ?")
```

```
Si (ok=1)
```

```
  $_s_entier1:=Demander("Entier 1";Chaine(ck_e_param1))
```

```
Fin de si
```

```
Si (ok=1)
```

```
  $_s_entier2:=Demander("Entier 2";Chaine(ck_e_param2))
```

```
Fin de si
```

```
Si (ok=1)
```

```
  ck_e_param1:=Num($_s_entier1)
```

```
  ck_e_param2:=Num($_s_entier2)
```

```
Fin de si
```

```
Jusque (ok=0)
```

```
$_e_Error:=JEX Free object ($_e_TargetObjetID) `libération de l'objet Java
```

Sinon

```
  ALERTE(JEX_ErrMsg ($_e_TargetObjetID))
```

Fin de si

```
ck_e_param1:=0
```

```
ck_e_param2:=0
```

CRÉATION DE L'OBJET

Nous venons d'exécuter une méthode de la classe cible afin de créer une instance de l'objet. Ce fonctionnement particulier à JExternal implique la création d'une méthode statique dédiée au sein du code de la classe Java :

Méthode statique de création d'objet

```

/*****
// createMyself - appelé par JEX Call object method
// spécifique JExternal pour identification de l'objet par un entier long
*****/
public static int createMyself()
{
    De obj= new De();
    int objectID = dbWorld.createObjectID( obj );
    return objectID;
}

```

La première ligne fait appel à une méthode dite **constructeur** chargée de créer et initialiser (ou encore instancier) l'objet : `De obj = new De() ;`

La seconde ligne utilise la méthode `createObjectID()` de la classe `DBWorld` du package `JExternal.jar` afin de récupérer un identifiant unique sur l'objet créé. Cet identifiant est ensuite retourné à 40 par la troisième ligne de code.

APPEL D'UNE MÉTHODE CALL-BACK

Il s'agit depuis l'exécution d'une méthode Java lancée par 4D, statique ou non, de rappeler une méthode 4D. Nous emploierons également ici la classe DBWorld du package JExternal.Jar.

La séquence consiste en :

- Appel dans 4D par la routine **JEX Call class method** de la méthode createDBWorld(). Création d'un objet DBWorld en lui passant le n° de process 4D à rappeler, celui-ci devant être le process de l'appel initial à la méthode Java par **JEX Call class method** ou **JEX Call object method**. Attention : cette étape est incontournable pour l'exécution de la suite de la méthode.
- Appel dans Java de la méthode **call4Dmethod** de l'objet DBWorld créé.

Exemple d'appel dans Java

Il s'agit d'appeler une méthode 4D pour provoquer l'affichage d'un dialogue de confirmation.

```
/******  
// add_LongIntVar_c3  
//callback simple  
/******  
public static int add_LongIntVar_c3( int param1, int param2 )  
{  
    String callbackMethodName="wrapper_confirm";  
  
    int result = param1+param2;  
  
    // appel callback à 4D  
    Object[] parameters = new Object[ 1 ];//parametres a passer a 4D  
    parameters[ 0 ] = new String( "Arrêter l'exécution ?" );  
    Object resultObj = null;  
  
    try{  
        resultObj = dbWorld.call4DMethod( "wrapper_confirm", parameters );  
    }  
    catch( InvalidMethodNameException e ){  
        DBWorld.alert( "An InvalidMethodNameException occured." );  
    }  
    catch( InvalidParametersException e ){  
        DBWorld.alert( "An InvalidParametersException occured." );  
    }  
  
    return result;  
}
```

DBWorld.alert() permet d'afficher simplement une alerte 4D depuis une méthode Java : très utile pour la gestion des erreurs et le débogage.

Il est cependant préférable de passer par une méthode "wrappante" (voir le glossaire) pour deux raisons :

- Lors de l'exécution sur une plateforme Windows, si la chaîne à afficher contient des caractères accentués, ils seront transmis en alphabet windows à la commande **ALERTE** de 4D. Celle-ci attend de l'alphabet MacOS quelle que soit la plateforme d'exécution. Le passage par une "wrappante" permet, le cas échéant, d'appliquer une conversion **Windows vers Mac** avant d'afficher l'alerte.
- Il est préférable de remonter à 4D des codes d'erreurs qui seront traduits en alertes (avec une éventuelle gestion de localisation) ou gérés différemment dans le cas d'une exécution sur 4D Server, par exemple en les journalisant dans un fichier.

Remarque sur la structure try-catch utilisée dans le code précédent.

En Java, les erreurs d'exécution sont appelées **exceptions**. Certaines méthodes sont susceptibles de remonter des erreurs au programme appelant. Leur déclaration est suivie du mot-clé **throws** puis du type d'exception générée. Ex. :

```
public DBWorld(int processNo) throws com.ronri_kobo.JExternal.InvalidProcessNoException
```

Tout appel à la méthode en question devra alors être encadré par du code chargé d'intercepter les exceptions générées : c'est le rôle de la structure try-catch.

```
    Try {  
        Appel pouvant générer une exception  
    }  
    catch (exception e) {traitement à appliquer à l'exception interceptée}
```

EXÉCUTION DE JAVASCRIPT

Il s'agit d'exécuter du code Javascript depuis 4D au moyen d'une librairie de code Java. Cette démonstration utilise une classe RunScript publiée sous licence publique GNU.

Remarque.

La licence publique GNU GPL (*General Public License*) est conforme à la définition Open Source définissant un logiciel "libre" : <http://www.opensource.org/licenses/gpl-license.html>

La méthode run() de RunScript permet d'exécuter un script passé sous forme de String Java (ce qui correspond à une chaîne de caractères).

Ronri Kobo a complété le package initial par une classe Access4D comprenant quelques méthodes permettant la communication avec 4D. Il est ainsi possible de rappeler 4D pour afficher un **Demander** ou un **CONFIRMER** via une méthode de rétro-appel, comme décrit dans le paragraphe précédent.

La méthode 4D permettant d'exécuter un Javascript est très simple :

Méthode d'exécution d'un javascript

```
`déclaration de variables  
C_ENTIER LONG($Err)
```

C_ENTIER LONG(vCurrentProcNo)

vCurrentProcNo:=**Numero du process courant**

`les paramètres à passer lors de l'appel de RunScript.run

TABLEAU POINTEUR(\$Parameters;2)

\$Parameters{1}:=>vJavaScript `contient le script à exécuter

\$Parameters{2}:=>vCurrentProcNo `nécessaire pour le callback

`exécution de RunScript.run

\$Err:=**JEX Call class method** ("RunScript";"run";\$Parameters)

Si (\$Err=0)

`Récupération du résultat dans vResult

Si (**Taille tableau**(\$Parameters)=1)

Si (**Type**(\$Parameters{1}->)=2) `Text

vResult:=vResult+\$Parameters{1}->+**Caractere**(ASCII CR)

\$Parameters{1}->:=""

Sinon

U_ERROR (-2;"JEX Call class method:Return type")

Fin de si

Sinon

U_ERROR (-1;"JEX Call class method:Return size")

Fin de si

Sinon

U_ERROR (\$Err;"JEX Call class method")

Fin de si

CONCLUSION

Ce plug-in sans équivalent apporte à 4D une ouverture considérable vers un secteur entier du développement logiciel.

Néanmoins, deux contraintes importantes sont à prendre en compte :

- la classe cible ne doit pas interagir directement avec l'utilisateur mais repasser systématiquement par 4D au travers d'un rétro-appel.
- l'absence de langage objet dans 4D impose l'emploi de classes utilitaires (DBWorld) et d'une méthode statique de création d'objet.

Pour interagir correctement avec les classes Java, il sera donc nécessaire :

- soit de modifier les classes que l'on souhaite réutiliser pour y ajouter ces éléments ;
- soit de développer une classe d'interfaçage qui recevra les appels de 4D et les transformera en appels Java vers la classe à réutiliser.

Tél.: 02 97 40 13 27

Site web : www.ckti.com

Contact : ckti@ckti.com

Lorsqu'on dispose des sources des classes à réutiliser, la première approche s'avère sans doute la plus souple.

AUTRES DÉMOS FOURNIES PAR RONRI KOBO

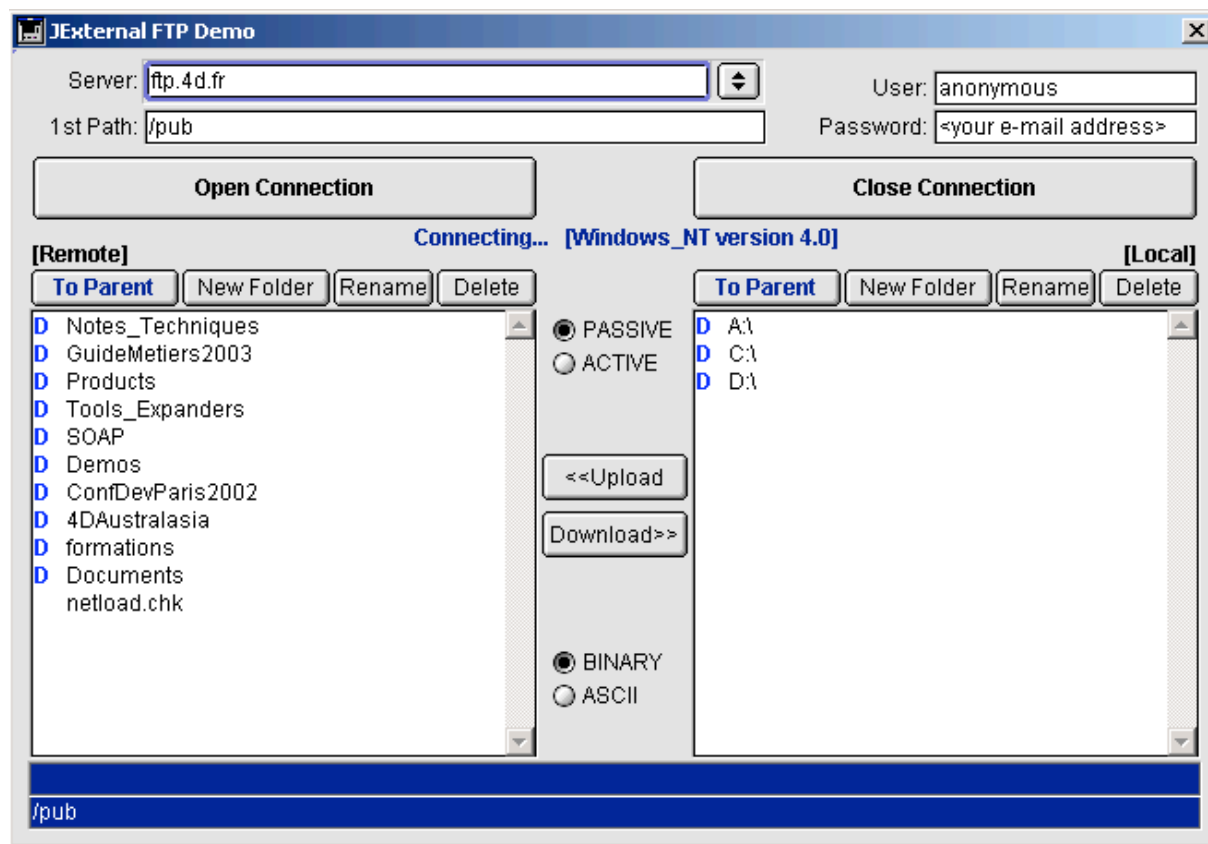
JEX_JDBCDEMO

Cette base permet la connexion à une base Oracle ou MS SQL Server 2000 via JDBC.

JEX_FTPDEMO

Cette base met en œuvre, comme son nom l'indique, un client FTP complet.

4D ne gère que la partie interface, toutes les commandes FTP sont gérées par des méthodes de classes Java.



Dialogue de connexion FTP utilisant JExternal <file>FTPDemo.tif</file>

CRÉATION DE PDF DEPUIS UN DOCUMENT XML

Cette démo, nommée JEX_PDF_FOPDemo, se révèle particulièrement intéressante car elle illustre ce que nous présentions comme primordial dans notre introduction : accéder aux plus récents développements autour d'XML, sans attendre que 4D les intègre en interne.

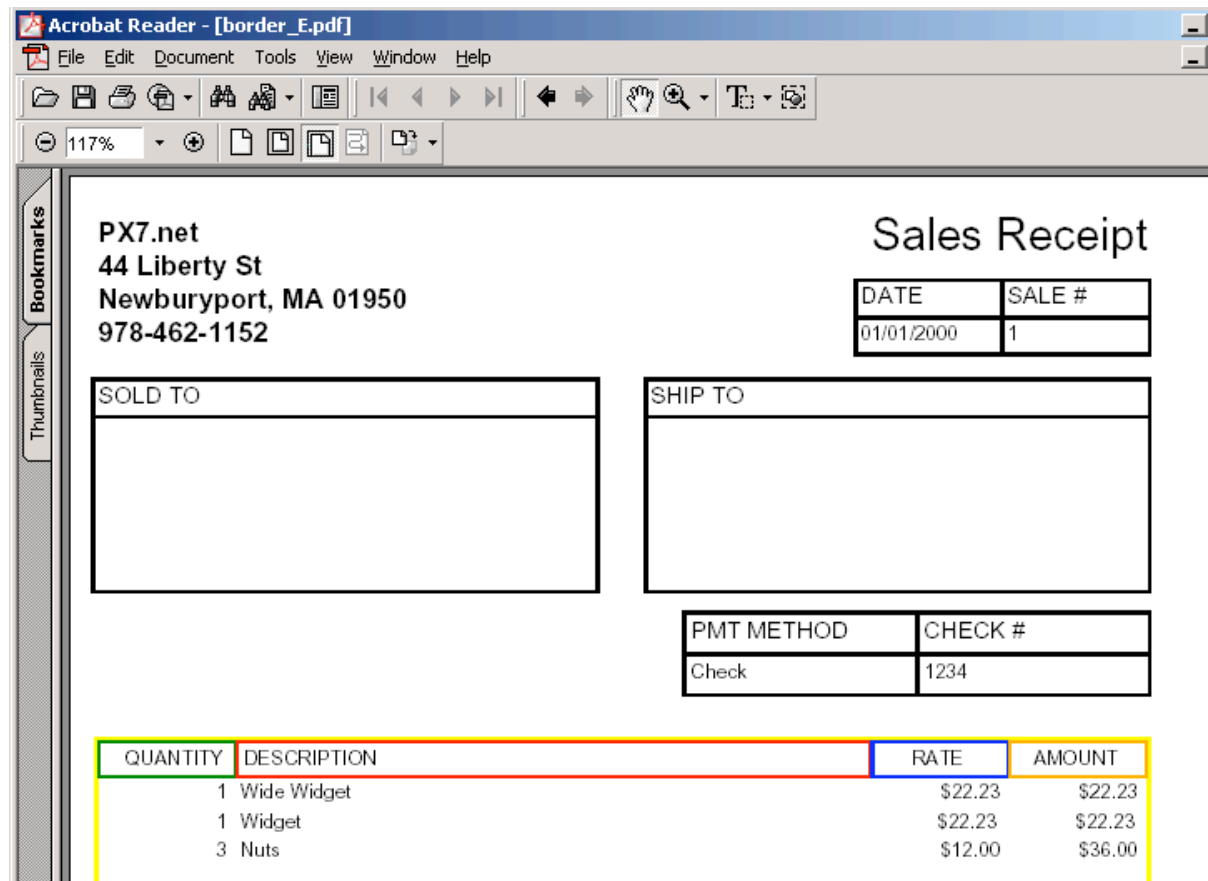
La base utilise FOP (*Formatting Objects Processor*), projet Apache de transformation d'un document XML en PDF, fondé sur la spécification XSL-FO : <http://xml.apache.org/fop/>.

Pour une présentation rapide de XSL-FO, se reporter aux sites :

Tél.: 02 97 40 13 27
 Site web : www.ckti.com
 Contact : ckti@ckti.com

- <http://tecfa.unige.ch/guides/tie/html/xml-xslfo/xml-xslfo.html>.
- <http://www.w3schools.com/xslfo/default.asp>

Ronri-Kobo a effectué un travail réellement impressionnant, permettant, en deux appels à des méthodes 4D, d'obtenir une représentation PDF d'un formulaire construit à partir de données XML :



Exemple de document PDF généré<file>FOP.tif</file>

Cette base de démonstration ne dispense pas, bien entendu, de l'apprentissage des technologies XSL-T et XSL-FO qui sont tout sauf intuitives. Néanmoins, cet outil facilite considérablement leur intégration dans 4D.

Les deux méthodes 4D principalement utilisées sont :

```
ErrorID :=FOP_CreatePDF (XMLBlobPtr ;PDFBlobPtr ;LogPtr ;LogLevel)
ErrorID :=FOP_LaunchPDF (PDFFilePath)
```

Il s'agit, dans le premier cas, de la "wrappante" d'un appel à la méthode createPDF de la classe com.ronri_kobo.fop.Fop4D en utilisant la routine de JExternal **JEX_Call_ClassMethod**.

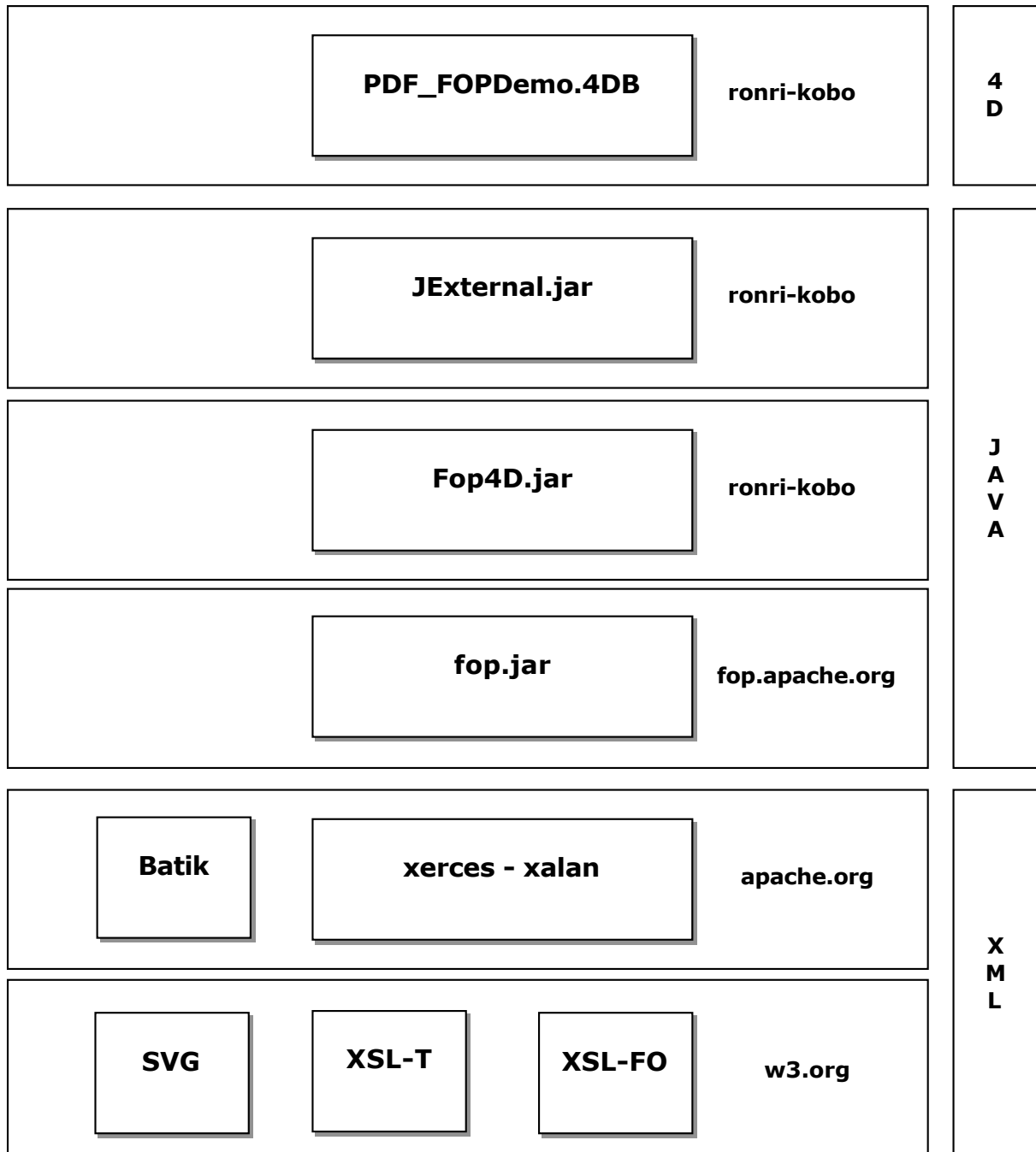
La classe com.ronri_kobo.fop.Fop4D sert d'interface aux classes open-source de fop.apache.org.

Le code source de Fop4D est fourni dans PDF_FOPDemo_*/javaSources/Fop4D.java.

Pour *FOP_LaunchPDF*, il s'agit de la "wrappante" d'un appel à `Runtime.getRuntime().exec ()`, une méthode classique de Java pour lancer des documents ou des applications en se détachant des particularités du multi plateforme.

Le code source est fourni dans `PDF_FOPDemo_*/JavaSources/Utilies.java`.

Pour apprécier la somme de travail requise et le degré de collaboration nécessaire à ce résultat, les couches technologiques mises en œuvre sont schématisées ci-dessous :



Architecture de la démonstration<file>FOP.xls</file>

Quelques précisions :

- **Xerces**: version java d'un *parser* XML issu d'un projet apache.org (c'est le parser utilisé par 4D dans 4D 2003) ;
- **Xalan**: version java d'un processeur de feuille de style XSL issu d'un projet apache.org ;

- **Batik** : trousse à outils java de manipulation d'images au format SVG (*Scalable Vector Graphics*), encore un projet apache.org.

Ce schéma ne mentionne pas le PDF d'Adobe qui représente bien sûr le maillon final indispensable à une présentation de qualité destinée à l'édition papier.

Remarque.

La mise à jour de Java 1.4.x doit être installée pour pouvoir utiliser pleinement la base de démonstration sur MacOSX.

AUTRES PLUG-INS DE RONRI-KOBO

La société ronri-kobo propose également **Jbyj** un autre plug-in de connectivité pour l'environnement 4D. Disponible pour MacOS 8/9 OSX et Windows, ce plug-in propose deux types d'appels de méthode projet 4D : depuis un autre 4D, mono, client ou serveur ; depuis une application Java (JSP, servlet, applet, application...).

Jbyj se base sur Java **RMI** (*Remote Method Invocation*) une technologie d'appel de méthodes à travers un réseau TCP/IP (à la manière de SOAP, le protocole des services Web). Cependant à la différence de SOAP qui repose sur HTTP, RMI utilise des connexions persistantes propriétaires tout comme un système client/serveur traditionnel. Tout au long d'une session, un client Jbyj connecté à un serveur Jbyj utilise donc le même process 4D, ce qui lui permet de réutiliser variables process et sélections et lui procure une meilleure réactivité qu'un service Web. Des routines offrent le transfert d'un enregistrement dans un Blob et inversement.

À noter:

- Jbyj ne nécessite pas une version serveur de 4D ;
 - Jbyj fonctionne dès la version 6.5.4 de 4D;
 - seule la version Pro de Jbyj permet d'appeler des méthodes 4D depuis une application Java.
-