

Approche de JNI via Delphi

par Philippe Sénéchal ([Accueil](#))

Date de publication : 26 août 2008

Dernière mise à jour :

Ce tutoriel a pour objectif de découvrir l'utilisation de JNI avec Delphi.

I - Avant propos.....	3
II - Pré-requis.....	3
III - Le programme java.....	3
IV - Création de la dll avec JavaToDPR.....	4
V - Implémentation de la fonction en Delphi.....	4
VI - Compilation du code source Delphi.....	6
VII - En conclusion.....	6
VIII - Remerciements.....	7

I - Avant propos

La portabilité est l'un des avantages souvent avancé dans le choix de Java en tant que langage de programmation. Toutefois, pour des raisons de communication avec d'autres programmes (ou le système d'exploitation), on est parfois appelé à développer une passerelle à l'aide d'autres langages. JNI a été intégré à la JRE dans ce but, mais son utilisation nécessite de maîtriser le C ou le C++. Il existe pourtant une possibilité sous Windows, d'utiliser JNI en programmant en Delphi, qui possède par ailleurs de nombreux composants de communication (COM, OLE, DDE, registre).

Cette solution est documentée par Matthew Mead à cette adresse [Using the Java Native Interface with Delphi](#), et repose sur l'utilisation de la bibliothèque `jni.pas` qui est une interface en Delphi de la Java Native Interface.

Programmant essentiellement en Java (et en C# qui en est assez proche), je me suis intéressé à cette solution à partir d'un besoin : j'ai programmé un utilitaire permettant de manipuler les données issues d'un autre programme développé sous Omnis 7.1. Ce dernier manipule un fichier dont les données sont stockées dans des tables au format Omnis. On peut en java accéder aux données de ces tables via le driver `OdbcJdbc`, mais pour savoir quelle est la fiche en cours d'utilisation, il faut utiliser le canal DDE (Dynamic Data Exchange).

DDE est un protocole client-serveur défini par Microsoft pour l'échange de données entre applications. DDE est désuet, mais reste utilisable pour beaucoup d'applications développées à l'époque où il était d'actualité. (Excel, Word, produit développé sous Omnis, 4D, et bien d'autres probablement...). Ne possédant pas de connaissance en C et C++, je pensais JNI hors de ma portée, jusqu'à ce que je découvre cette possibilité en Delphi. Je vais donc développer l'abord de JNI avec Delphi à partir de la construction d'une dll d'accès aux données DDE.

II - Pré-requis

Delphi peut être téléchargé en version gratuite : [Télécharger et installer Turbo Delphi Explorer](#)
`jni.pas` (et `JNI_MD.INC`) peuvent être téléchargés ici : [jni_pas.zip](#)
JavaToDPR peut être téléchargé ici : [JavaToDPR.zip](#)

III - Le programme java

J'utilise ici une classe minimaliste, avec une méthode `main` permettant d'afficher sur la console l'index de la fiche que l'on va rechercher sur le canal DDE.

La méthode qui permet cette recherche se nomme `getNCE()` ; elle doit retourner un entier. Elle est précédée du mot clé 'native' et n'a pas de corps : celui-ci sera programmé dans la dll.

Enfin, il y a un bloc `static` qui crée le lien avec la dll.

code Java : 'DDEToJava.java'

```
public class DDEToJava {  
  
    public static void main(String[] args) {  
        System.out.println(getNCE());  
    }  
  
    native public static Integer getNCE();  
  
    static {  
        System.loadLibrary("DDEToJava");  
    }  
  
}
```

IV - Création de la dll avec JavaToDPR

Une fois compilée la classe DDEToJava, on peut utiliser le programme JavaToDPR avec la commande :

en ligne de commande :

```
java JavaToDPR -o DDEToJava.dpr DDEToJava
```

Cette commande va créer un fichier DDEToJava.dpr que l'on va pouvoir ouvrir avec Delphi.

JavaToDPR permet de respecter la syntaxe utilisée dans jni.pas.

Le nom de notre fonction native a comme syntaxe : Java_NomDeLaClasse_NomDeLaMéthode.

Elle a 2 arguments propres :

- un Objet PJNIEnv qui correspond à un environnement JNI et permet d'accéder à des fonctions qui implémentent les opérations de base de l'interopérabilité avec Java. Cet environnement JNI contient des informations sur la zone de mémoire où la machine virtuelle de Java est allouée, le fil d'exécution (thread) sur lequel l'environnement JNI est créé, et des pointeurs vers les fonctions qui implémentent les opérations de la JNI.
- et un Objet JObject équivalent de 'this' en java.

code Delphi : 'DDEToJava.dpr'

```
// Il s'agit d'une dll
library DDEToJava;

// importe jni.pas
uses JNI;

// fonction qui retourne un entier JInt
function Java_DDEToJava_getNCE (PEnv: PJNIEnv; Obj: JObject ): JInt; stdcall;
// Le corps de la fonction est vide
begin
end;

exports
    // déclare la fonction
    Java_DDEToJava_getNCE;

end.
```

V - Implémentation de la fonction en Delphi

Pour implémenter cette fonction, on va avoir besoin du composant 'DdeClientConv' qui appartient à la bibliothèque 'DdeMan'.

Ce composant représente une conversation DDE avec une application serveur DDE ; il est accessible à partir de l'onglet 'système' dans la barre des composants de Delphi. Comme on crée une dll sans fenêtre, on ne va pas se servir du composant visuel, mais on va l'instancier avec la variable 'Nil'.

Puis on spécifie le service et la rubrique de la conversation DDE : ici c'est AXILOG (Le programme s'appelle Axilog.exe) et AXISANTE (la librairie utilisée par le programme se nomme : Axisante.lbr). Ce point nécessite de connaître des données propres aux programmes utilisés.

Et à l'aide de ce lien, on va chercher la valeur souhaitée : ici, c'est la valeur contenue dans 'NCEPatient' correspondant au nom de la clé primaire de la table traitant les fiches. Une conversion est nécessaire, car on récupère une chaîne, et la fonction retourne un entier.

Ce qui nous donne le listing final :

code Delphi : 'DDEToJava.dpr'

```

library DDEToJava;

// on rajoute la bibliothèque utilisée
uses JNI, DdeMan;

function Java_DDEToJava_getNCE (PEnv: PJNIEnv; Obj: jobject ): jint; stdcall;
begin
    // on initialise la variable de retour
    result := 0;
    // instantiation de la conversation DDE
    with TDdeClientConv.Create(nil) do
    try
        // connexion manuelle
        ConnectMode := ddeManual;
        // on crée le lien avec les variables du canal DDE
        SetLink('AXILOG','AXISANTE');
        // on teste ce lien
        if OpenLink then
            // si tout s'est bien passé, on renvoie la valeur contenue dans la variable qui nous intéresse
            result := StrToInt(RequestData('NCEPatient'));
        finally
            // On libère les ressources utilisées
            Free;
    end;
end;

exports
    Java_DDEToJava_getNCE;

end.
    
```

Pour utiliser cette fonction d'une façon plus générique, il suffit de mettre les 3 variables (String) utilisées (AXILOG,AXISANTE, NCEPATIENT) en argument de la fonction. Cette approche soulève l'une des difficultés de l'utilisation de JNI, à savoir la correspondance entre les types de données de chaque langage (types de base , String ...).

Si Java n'a qu'un format de String, Delphi utilise lui 3 formats différents :

Type	Longueur maximum	Mémoire nécessaire	Utilisation
ShortString	255 caractères	de 2 à 256 octets	Compatibilité ascendante
AnsiString	~2 ³¹ caractères	de 4 octets à 2Go	Caractères sur 8 bits (ANSI), DBCS ANSI, MBCS ANSI, etc
WideString	~2 ³⁰ caractères	de 4 octets à 2Go	Caractères Unicode ; serveurs multi-utilisateurs et applications multilingues

Pour pouvoir effectuer cette correspondance pour les String qui nous intéressent ici, il va donc nous falloir créer une instance de l'objet TJNIEnv, puis utiliser la fonction de conversion fournie par cet objet : JStringToString, ce qui modifie notre code source en :

code Delphi : 'DDEToJava.dpr'

```

library DdeToJava;

uses JNI,DdeMan;

function Java_DdeToJava_getNCE (PEnv: PJNIEnv; Obj: jobject; Arg1 : JString; Arg2 : JString; Arg3 : JString): jint; stdcall;
// Arg1 = service DDE Arg2 = rubrique DDE Arg3 = data
var
    
```

code Delphi : 'DDEToJava.dpr'

```
// Utilisation d'une variable TJNIEnv
JVM: TJNIEnv;
begin
  // instantiation de l'environnement JNI
  JVM := TJNIEnv.Create(PEnv);
  Result := 0;
  with TDdeClientConv.Create(nil) do
  try
    ConnectMode := ddeManual;
    // on utilise les arguments après conversion en chaîne Delphi
    SetLink(JVM.JStringToString(Arg1),JVM.JStringToString(Arg2));
    if OpenLink then
      Result := StrToInt(RequestData(JVM.JStringToString(Arg3)));
  finally
    Free;
    JVM.Free;
  end;
end;

exports
  Java_DdeToJava_getNCE;

end.
```

Il faut bien entendu ne pas oublier de modifier la fonction native du fichier Java, en rajoutant nos 3 arguments chaînes :

code Java : extrait de 'DDEToJava.java'

```
native public static Integer getNCE(String s1, String s2, String s3);
```

VI - Compilation du code source Delphi

Il ne nous reste plus qu'à compiler ce code source pour avoir une dll utilisable. Pour la compilation, il est nécessaire que le fichier 'DDEToJava.dpr' soit dans le même répertoire que 'jni.pas' et 'JNI_MD.INC'. On compile à l'aide du menu 'Projet', puis 'compiler DDEToJava'.

On a alors notre dll qu'il faut placer dans le répertoire System32 de Windows, avant d'exécuter le programme DDEToJava.java . Une solution plus intéressante si ce n'est pas une dll partagée, consiste à la placer dans le répertoire du programme java et de modifier l'appel à la dll comme suit :

code Java : extrait de 'DDEToJava.java'

```
System.load(System.getProperty("user.dir")+File.separator+"DDEToJava.dll");
```

VII - En conclusion

Cet exemple a permis de découvrir la mise en oeuvre de JNI, à travers un besoin bien spécifique, en utilisant Delphi. La documentation de Matthew Mead sur ce sujet est assez complète, et peut permettre de résoudre des situations bien plus complexes.

Sources

Using the Java Native Interface with Delphi - Documentation de l'utilisation de jni.pas de Matthew Mead

Penser en Java de Bruce Eckel - chapitre sur JNI

Un cadre conceptuel pour l'interopérabilité des langages de programmation- thèse de Gustavo A. Ospina Agudelo

Utilitaires DDE

DDE For Java de JavaParts - librairie shareware d'accès à DDE en Java

DDE Monitor de Java Parts- utilitaire de monitoring des conversations DDE (free version)

VIII - Remerciements

Je remercie **Ricky81** qui m'a guidé pour la publication de cet article, **Nono40** et **Pedro** pour l'optimisation du code Delphi et **Diogène** pour ses corrections orthographiques.