Developpez.com Club des développeurs

Réaliser un plug-in comportant un composant

par Sébastien Doeraene (sjrd.developpez.com) Éric Leconte (clorish.free.fr)

Date de publication : 10/11/2004

Dernière mise à jour : 07/02/2005

Réaliser un plug-in en Delphi dans lequel est définit un composant qui doit être inséré dans une fiche de l'application.

Introduction
I - Utilisation des paquets Borland
I-A - Les paquets
I-A-1 - Définition
I-A-2 - Pourquoi les paquets et leurs modifications ?
I-A-3 - Les paquets remplacent-ils les DLL ?
I-B - Utiliser un paquet
I-B-1 - Chargement et déchargement
I-B-2 - Charger une classe et l'instancier
I-B-3 - Conclusion
I-C - Créer le plug-in
I-C-1 - Mise en place d'un programme de test
I-C-2 - Code du composant plug-in
I-C-3 - Créer le paquet
I-C-3.1 - Créer un nouveau paquet
I-C-3.2 - Ajout de l'unité uMonPanel
I-C-3.3 - Compilation du paquet
I-D - Une classe de chargement générique
I-D-1 - Code complet
I-D-2 - Explications du code
I-D-2.1 - Chargement/Déchargement
I-D-2.2 - InstanceList
I-E - Aller plus loin
I-F - Conclusion
II - Utilisation des interfaces
II-A - Un exemple concret
II-B - Déclaration de l'interface
II-B-1 - L'indispensable dans l'unité
II-B-2 - Méthodes additionnelles de l'interface
II-B-3 - Interfaces optionnelles
II-B-3.1 - Ajout de commandes du plug-in dans le menu de la fenêtre de l'application
II-B-3.2 - Redimensionnement du composant
II-A-3.3 - Evénements de l'application que le plug-in veut intercepter
II-B-4 - Code complet de l'unité PluginIntf
II-C - Utilisation côté DLL/paquet
II-C-1 - Code complet de l'unité TestPlugin
II-C-2 - Explications du code
II-D - Utilisation côté application
II-D-1 - Classe TPlugin
II-D-1.1 - Code complet de l'unité PluginClass
II-D-1.2 - Etude du code
II-D-1.2a - Méthode CreateInstance
II-D-1.2b - Méthode ReleaseInstance
II-D-1.2c - Méthodes AddCommand et AddCommands
II-D-1.2d - Méthodes Load et Unload
II-D-2 - Le code de la fenêtre principale
II-E - Conclusion
Remerciements

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Introduction

Nombreux sont les développeurs Delphi qui ont eu ce problème : celui d'insérer un composant définit dans une dll/un paquet dans une fiche de l'application. Il existe plusieurs solutions à ce problème. Deux d'entre elles vont vous être exposées ici. La première, basée sur les paquets Borland, est l'idée de Clorish. Cette méthode a le mérite d'être simple à mettre en oeuvre, tant que l'on a pas besoin de communication importante avec le plug-in. La seconde, basée sur les interfaces, est l'idée de sjrd. Celle-ci est beaucoup plus longue et complexe à réaliser, mais permet une énorme souplesse de communication.

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

I - Utilisation des paquets Borland

Voici la première solution proposée pour résoudre ce problème. L'idée est d'utiliser le mécanisme des paquets Borland et les informations de types à l'exécution (RTTI). Les méthodes à ne pas louper sont **RegisterClass**/**UnregisterClass** et **GetClass**/**FindClass**. Cette solution est simple et rapide, et ne génère pas de complications. Elle est cependant limitée au niveau communication avec le plug-in : si vous devez communiquer avec le plug-in, je vous recommande d'utiliser la seconde méthode. En effet, si vous avez besoin de communication, soit de méthodes, spécifique, vous devrez créer un parent commun à toutes les instances de plug-in et y définir ces méthodes en **virtual** voire **abstract**, ce qui empêche les créateurs des plug-in de créer ceux-ci à partir d'un autre composant.

I-A - Les paquets

I-A-1 - Définition

Un paquet n'est rien d'autre qu'une simple DLL que Borland a améliorée en incluant les Informations de Types à l'Exécution (RTTI : RunTime Type Informations) qui manquaient aux DLL.

I-A-2 - Pourquoi les paquets et leurs modifications ?

Tout d'abord, le format des DLL a été mis en place à l'époque ou la programmation Objet n'existait pas encore, ce qui aujourd'hui n'en fait pas le format le plus adapté pour exporter des objets. De plus, les structures Objets définies sous Delphi et C++ ne sont pas compatibles, ce qui rend encore plus difficile l'exportation d'objets depuis des DLL pouvant être chargées aussi bien depuis des applications Delphi que depuis des applications C++.

De là l'avantage des nouvelles DLL de la plateforme .NET, car leur structure est identique quel que soit le langage d'origine.

Si vous préférez .NET, consultez aussi ***ce tutoriel*** *réalisé par* ***morpheus****.*

I-A-3 - Les paquets remplacent-ils les DLL ?

Non, pas tout a fait. DLL et paquets jouent des rôles différents et plutôt complémentaires.

Borland a mis en place les paquets dans un premier temps à usage interne pour la gestion des composants dans ses EDI, comme Delphi.

Delphi, dans ses versions 1 et 2, gérait sa palette de composants en écrivant le code de ces derniers dans une DLL. Ce système avait ses limites : Le fonctionnement de chargement des DLL empêchait l'éditeur de charger dynamiquement ses composant en cours de session de programmation. Toute la palette était présente dans l'EDI. De plus, pour ajouter ses propres composants, il fallait ajouter son code a celui de la DLL, la recompiler, redémarrer l'EDI pour charger les nouveaux composants, etc...

Pour simplifier ces opérations, Borland a amélioré le format des DLL pour résoudre ces problèmes, ce qui a donné naissance a ce que nous appelons des « paquets ».

I-B - Utiliser un paquet

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

La gestion statique d'un package (1) est très simple à mettre en place mais à mon goût manque beaucoup de sécurité dans le cas de mises a jours des paquets, je déconseille donc cette méthode tant que les paquets doivent évoluer. Je parlerai donc uniquement de la gestion **dynamique**.

I-B-1 - Chargement et déchargement

Un package étant une DLL améliorée, on retrouve beaucoup de similitudes quant au chargement/déchargement (2) .

Charger et décharger un package

```
Var Module : THandle;
...
Module := LoadPackage('C:\MonPaquet.bpl'); // Chargement
...
UnloadPackage(Module); // Déchargement
```

I-B-2 - Charger une classe et l'instancier

Ensuite, le chargement d'une classe se fait par l'intermédiaire d'une fonction très pratique : **GetClass** (voir aussi **FindClass**).

Pour reconnaître les erreurs relatives aux plug-in des autres, nous allons déclarer un nouveau type d'erreur.

Charger une classe exportée par un package et l'instancier

```
Type
  EPluginError = Class(Exception);

...

Var MaClasse : TPersistentClass; // Classe de base
    Instance : TPersistent;      // Instance de l'objet a charger
...
MaClasse := GetClass('TMaClasse'); // Chargement de la classe depuis le package
If MaClasse = nil Then // GetClass renvoie nil en cas d'erreur
  Raise EPluginError.CreateFmt('Classe ''%s'' non trouvée', ['TMaClasse']); // Erreur
// Succes :
Instance := MaClasse.Create; // Création de l'objet
```

Et voila ! Ce n'est pas très compliqué. Le principe repose sur l'utilisation des **métaclasses** plus communément appelées **types référence de classe**. Pour plus de détails sur les métaclasses, consulter le tutoriel **Références de classe ou métaclasses** par **Laurent Dardenne**.

Bon les choses ne s'arrêtent malheureusement pas là... ça serait trop beau. **MaClasse** est une variable de type **TPersistentClass**, c'est-à-dire un descendant direct de **TObject**. On constate que comme son parent, elle ne possède pas de constructeur **Create(AOwner : TComponent)** nécessaire a la création de composants graphiques tels que panels, boutons, etc...

Rien de grave, le système de transtypage est là (seule la dernière ligne change) :

Transtypage de l'instance créée

```
Var Classe : TPersistentClass; // Classe de base
    Instance : TPersistent;    // Instance de l'objet a charger
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Transtypage de l'instance créée

```
...
Classe := GetClass('TMaClasse'); // Chargement de la Classe depuis le package
If Classe = nil Then // GetClass renvoie nil en cas d'erreur
  Raise Exception.Create('Classe non trouvée'); // Erreur
// Succes :
Instance := TComponent(Classe).Create(Form1); // Création de l'objet
```

Bon là encore certains me diront : « C'est bien... mais bon, il va falloir tout le temps transtyper notre objet **Instance** pour modifier ses propriétés ? »

Bien sûr que non ! Il suffit de stocker la variable créée dans une variable de classe enfant, descendante de **TPersistent**.

Stocker l'instance sous une variable de classe enfant

```
Var Instance : TPanel;
...
Instance := TPanel(Classe).Create(Form1); // Création de l'objet
```

Bien sûr, le transtypage a toujours ses dangers. Que se passerait-il si la classe chargée dans **Classe** n'est pas un descendant de **TPanel** mais de **TButton** ? On peut heureusement contrôler toutes les informations de l'objet grâce aux informations de type RTTI.

Vérifier l'héritage d'une classe grâce aux RTTI

```
// Chargement de Classe
If not Classe.InheritsFrom(TPanel) Then
  Raise Exception.Create('Cast impossible');
// Création de l'instance
```

Ainsi on est sûr que la classe est bien un descendant de **TPanel**, donc le cast est possible et sans danger.

Voici un exemple concret et récapitulatif qui charge, crée, et gère un objet de type **TPanel** :

Charger, créer et gérer un panel

```
Var MonPanel : TPanel;

Procedure CreatePanel;
Var Classe : TPersistentClass;
    Instance : TPanel;
Begin
  Classe := GetClass('TMaClasse');
  If Classe = nil Then
    Raise Exception.Create('Classe non trouvée');
  If not Classe.InheritsFrom(TPanel) Then
    Raise Exception.Create('Cast impossible');
  MonPanel := TPanel(Classe).Create(Form1); // Création de l'objet
  With MonPanel do
  Begin
    Parent := Form1;
 Color := clPurple;
 Align := alClient;
  End;
End;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

I-B-3 - Conclusion

Une fois chargé, le package met a notre disposition toutes les classes qu'il contient. Après création du composant, son utilisation est exactement la même que lorsque l'on crée un exécutable seul.

I-C - Créer le plug-in

Maintenant que l'on a vu comment charger un composant dynamiquement depuis un paquet, regardons de plus près comment créer ce paquet de plug-in. Pour plus de facilités, nous ne présenterons la méthode que pour des paquets ne contenant qu'un seul plug-in ; mais il est tout à fait possible de créer des paquets contenant plusieurs plug-in, pour autant que le programme le sache (je passerai les détails).

I-C-1 - Mise en place d'un programme de test

Dans un premier temps, il est préférable de tester son composant dans un logiciel de test très simple :

- Une form
- Un bouton

On placera le code de création dynamique du composant dans l'événement **OnClic** du bouton.

I-C-2 - Code du composant plug-in

Le code de notre composant sera bien sûr dans une unité à part. De plus ce code ne doit pas référencer d'objets de l'exécutable.

Unité uMonPanel - Code du composant contenu dans le package

```
Unit uMonPanel;

Interface

Uses Sysutils, ExtCtrls, Classes;

Type
  TMonPanel = Class(TPanel)
  Public
    Constructor Create(AOwner : TComponent); Override;
  End;

Implementation

Constructor TMonPanel.Create(Aowner : Tcomponent);
Begin
  Inherited Create(Aowner);
  Self.Color := clPurple;
End;

Initialization
  RegisterClass(TMonPanel);
Finalization
  UnregisterClass(TMonPanel);
End.
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

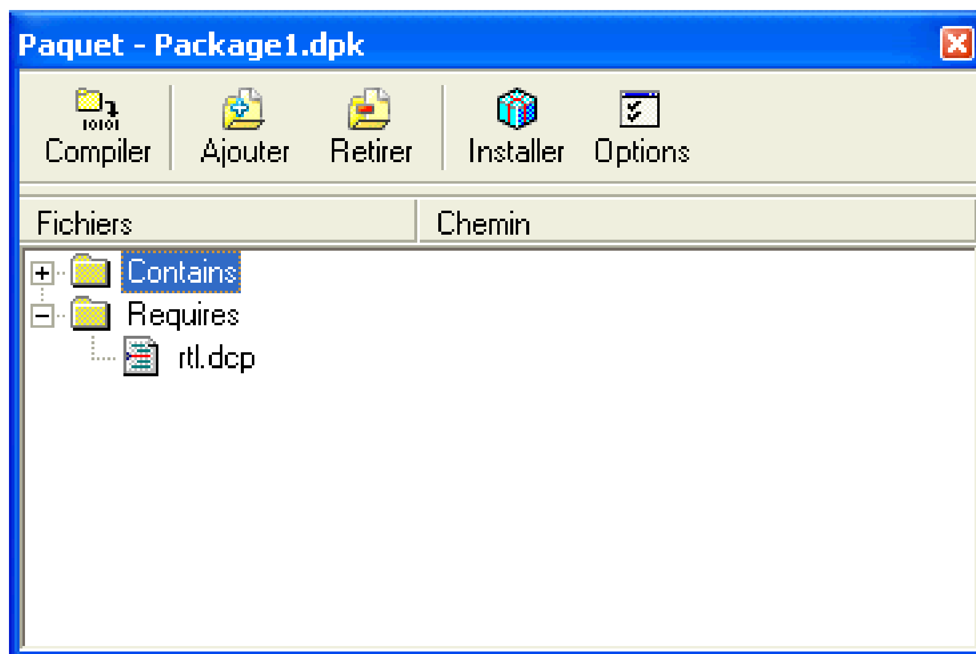
I-C-3 - Créer le paquet

Une fois notre unité créée, nous allons l'inclure dans un paquet. Pour ça, il faut ouvrir le code source d'un paquet existant, ou en créer un nouveau.

I-C-3.1 - Créer un nouveau paquet

Dans Delphi, fermez tous les projets en cours puis sélectionnez le menu **Fichier|Nouveau|Autre...** et dans l'onglet **Nouveau**, sélectionnez **Paquet**.

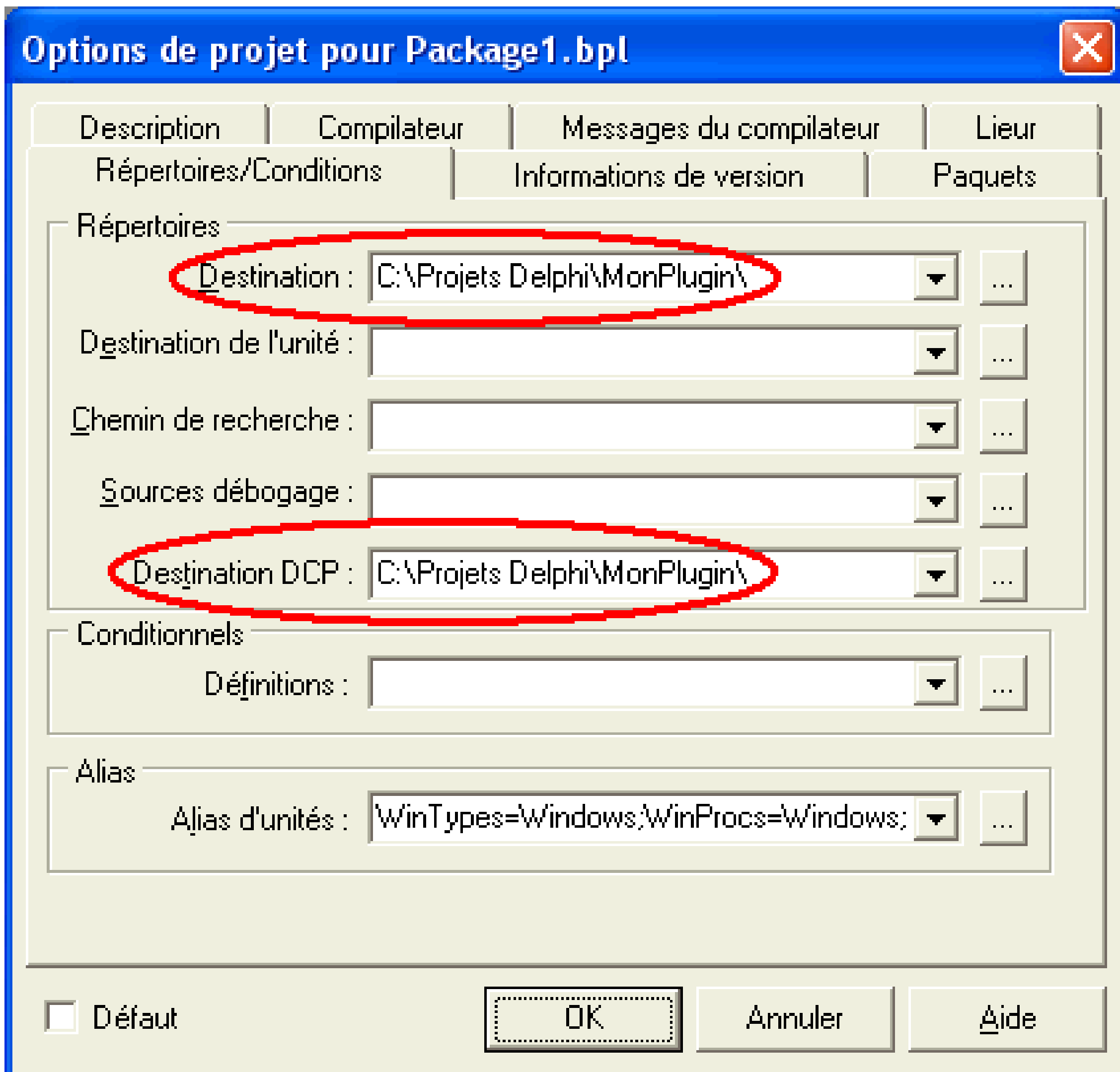
Vous pouvez voir apparaître une fenêtre de ce genre :



Maintenant plusieurs étapes sont à suivre. Commencez par sélectionner le menu **Projet|Options** ou cliquez sur le bouton **Options** de la fenêtre précédemment créée.

Sélectionnez l'onglet **Répertoires/Conditions** :

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

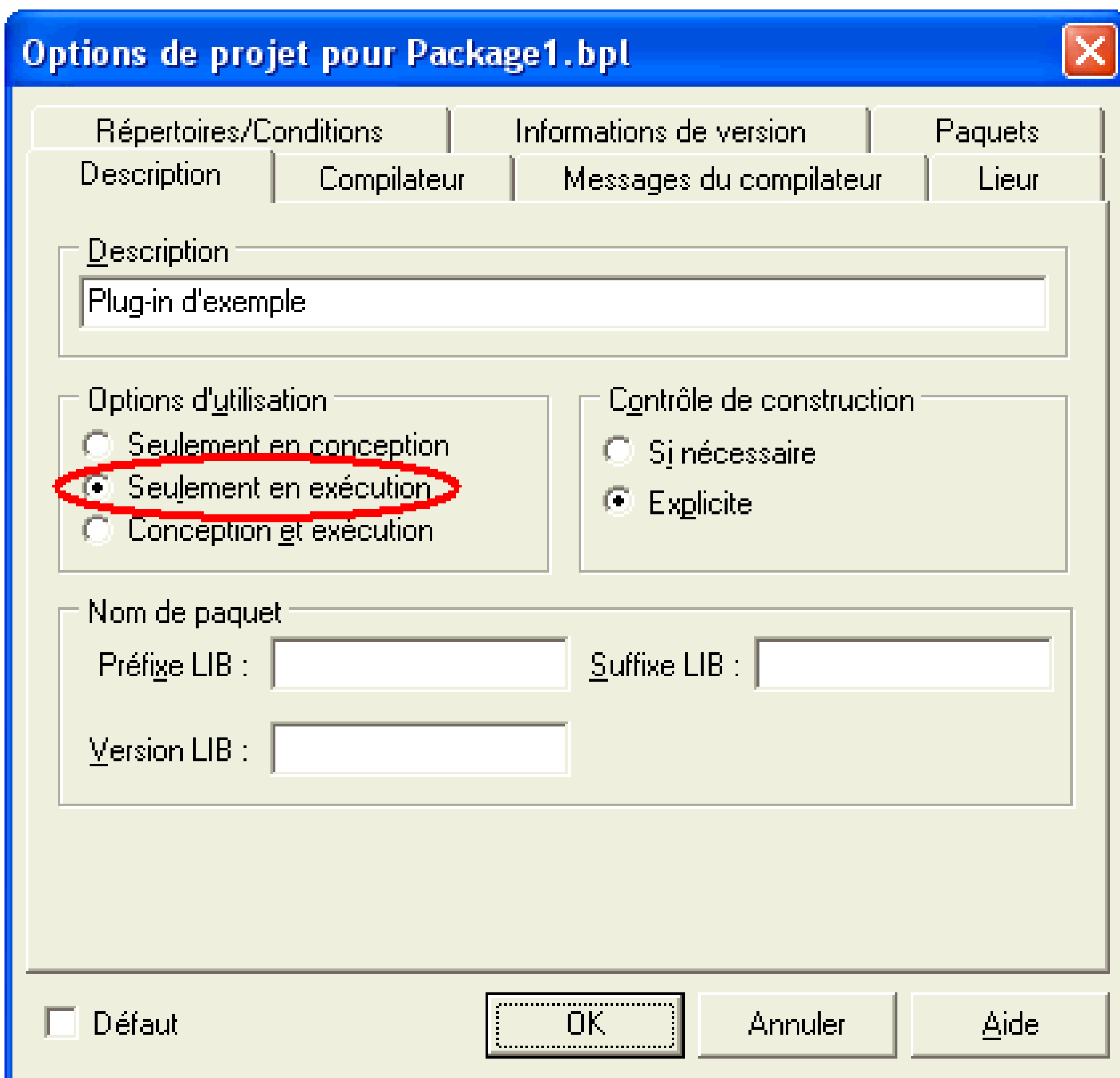


Destination : Chemin d'accès du répertoire où écrire le fichier .bpl (3)

Destination DCP : Chemin d'accès du répertoire où écrire le fichier .dcp (4)

Ouvrez maintenant l'onglet **Description** :

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.



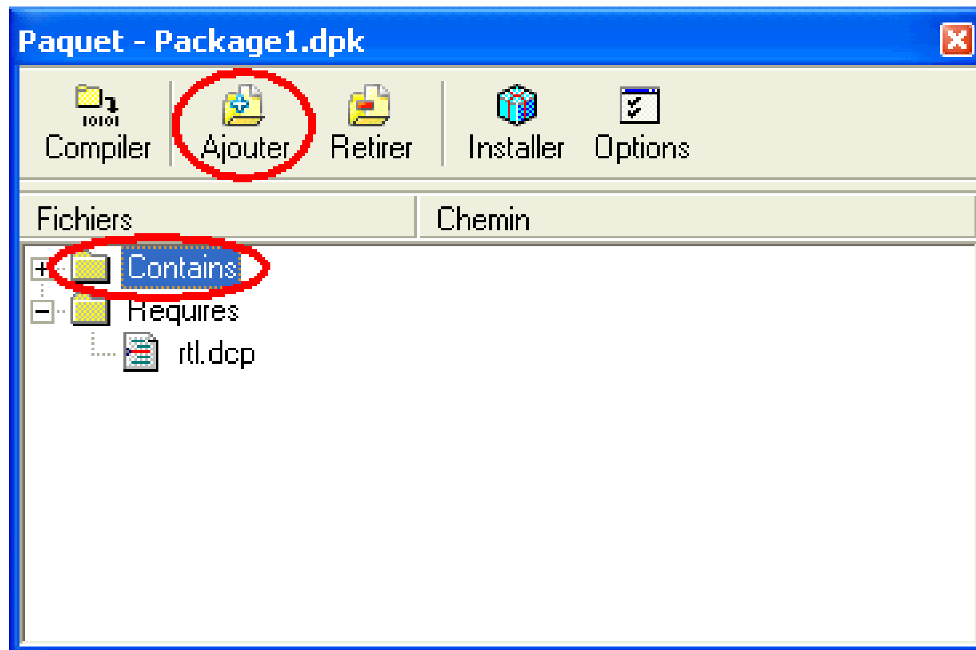
Options d'utilisations : Définir si notre paquet doit être en mode conception, exécution ou les deux. Dans le cas d'un paquet de plug-in, il faut choisir le mode **exécution** ; on ne désire en effet pas le charger dans l'EDI de Delphi.

Pour les autres onglets, je vous laisse fouiller, ils ne sont pas nécessaires dans notre cas.

I-C-3.2 - Ajout de l'unité uMonPanel

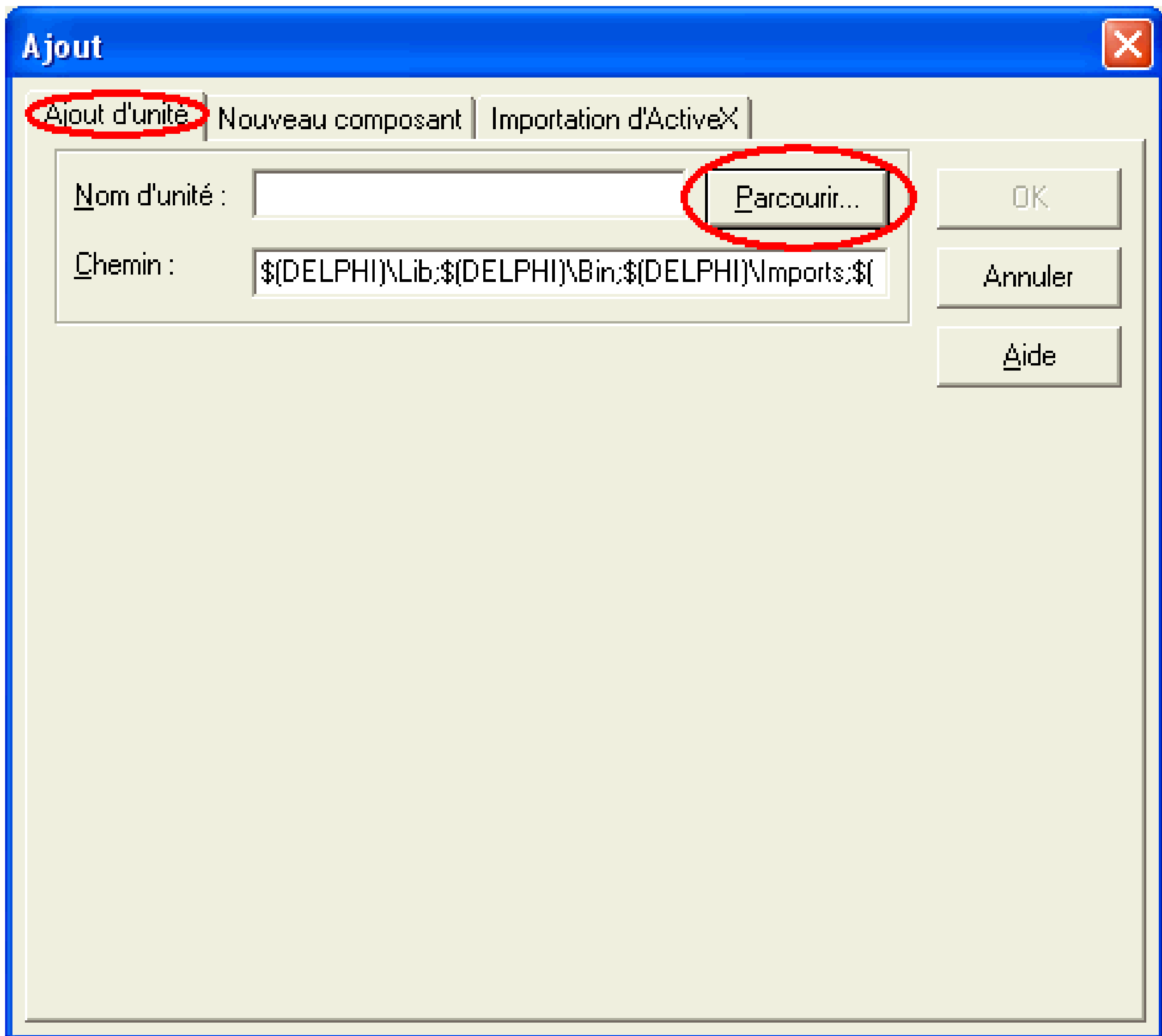
Sélectionnez la section **Contains** puis cliquez sur **Ajouter**.

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.



Une fenêtre ressemblant à ceci s'affiche :

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

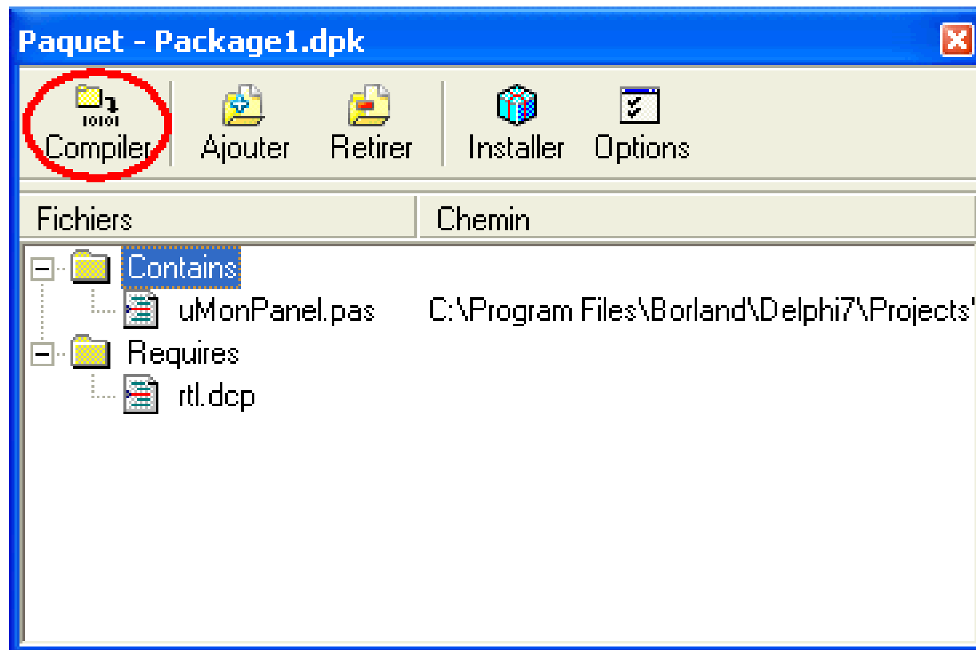


Dans l'onglet **Ajout d'unité**, cliquez sur **Parcourir** pour sélectionner le ou les fichiers souhaités. Ici sélectionnez l'unité **uMonPanel.pas** créée précédemment.

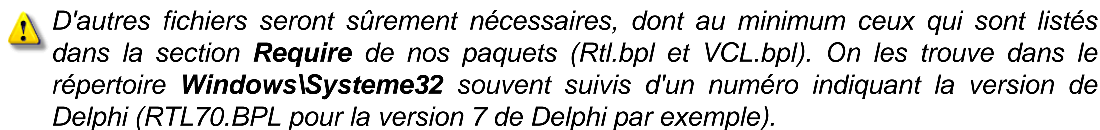
I-C-3.3 - Compilation du paquet

Une fois l'unité ajoutée, cliquez sur **Compiler**.

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.



Maintenant il ne reste plus qu'à récupérer le fichier .bpl (dans le répertoire spécifié dans les options) puis de le distribuer avec notre exécutable.



*D'autres fichiers seront sûrement nécessaires, dont au minimum ceux qui sont listés dans la section **Require** de nos paquets (Rtl.bpl et VCL.bpl). On les trouve dans le répertoire **Windows\Systeme32** souvent suivis d'un numéro indiquant la version de Delphi (RTL70.BPL pour la version 7 de Delphi par exemple).*

Le mieux est de faire tourner son application sur un poste où Delphi n'a pas été installé et de lister les paquets demandés.

I-D - Une classe de chargement générique

Nous allons créer un **"Manager de paquets"**, c'est-à-dire une classe qui s'occupe de la gestion mémoire des paquets et du chargement des classes ainsi que de la création des instances d'objets.

Cette classe a été écrite dans le but de montrer toutes les étapes de base de la gestion de classes depuis un paquet chargé dynamiquement. Elle peut être réutilisée pour la plupart des cas, même si elle est encore loin d'être universelle.

I-D-1 - Code complet

Unité pkgCommon - manager de paquets

```
unit pkgCommon;

interface

Uses SysUtils, Types, Windows, Dialogs, Classes, Controls, Contnrs;

Type
  TPkgManager = Class(TObject)
  Private
    pInstanceList : Array of TObject;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité pkgCommon - manager de paquets
```
    pPkgHandle : HModule;
    Procedure AddInstance(AObject : TObject);
    Procedure RemoveInstance(Index : Integer);
    Procedure ClearInstanceList;
    Procedure UpdateInstanceList;
  Public
    Destructor Destroy; Override;
    Procedure LoadPkg(APackage : String);
    Procedure UnLoadPkg;
    Function CreateInstanceOf(AClass : String) : TPersistent; Overload;
    Function CreateInstanceOf(AClass : String; AOwner : TComponent) : TControl; Overload;
  end;

implementation

ResourceString
  PKGALREADYCHARGED_ERROR_MSG = 'Un paquet est déjà chargé';
  LOADPKG_ERROR_MSG = 'Impossible de charger le paquet ''%s''';
  GETCLASS_ERROR_MSG = 'Impossible de charger la classe ''%s''';

Type
  TPkgGetClasses = Function : TStringDynArray;

Var
  GetClasses : TPkgGetClasses;

//--------------------------------------------------------------------------------

Destructor TPkgManager.Destroy;
Begin
  Self.UnLoadPkg;     // Appel direct car UnLoadPkg est une méthode "Safe"
  Inherited Destroy;
end;

//--------------------------------------------------------------------------------

Procedure TPkgManager.LoadPkg(APackage : String);
Begin
  If Self.pPkgHandle <> 0 then
    Raise Exception.Create(PKGALREADYCHARGED_ERROR_MSG);
  Self.pPkgHandle := LoadPackage(APAckage);
  If Self.pPkgHandle = 0 then
    Raise Exception.CreateFmt(LOADPKG_ERROR_MSG, [APackage]);
End;

//--------------------------------------------------------------------------------

Procedure TPkgManager.UnLoadPkg;
Begin
  If Self.pPkgHandle <> 0 Then
  Begin
    Self.ClearInstanceList;
 UnLoadPackage(Self.pPkgHandle);
  end;
End;

//--------------------------------------------------------------------------------

Function TPkgManager.CreateInstanceOf(AClass : String) : TPersistent;
Var Classe : TPersistentClass;
    Instance : TPersistent;
Begin
  Classe := GetClass(AClass);
  If Classe = Nil Then
    Raise Exception.CreateFmt(GETCLASS_ERROR_MSG, [AClass]);
  Instance := Classe.Create;
  Self.AddInstance(Instance);
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité pkgCommon - manager de paquets

```
    Result := Instance;
  End;

  //---------------------------------------------------------------------------------

  Function TPkgManager.CreateInstanceOf(AClass : String; AOwner : TComponent) : TControl;
  Var Classe : TPersistentClass;
      Instance : TControl;
  Begin
    Classe := GetClass(AClass);
    If Classe = Nil Then
      Raise Exception.CreateFmt(GETCLASS_ERROR_MSG, [AClass]);
    Instance := TControlClass(Classe).Create(AOwner);
    Self.AddInstance(Instance);
    Result := Instance;
  End;

  //---------------------------------------------------------------------------------

  Procedure TPkgManager.AddInstance(AObject : TObject);
  Var Size : Integer;
  Begin
    Size := Length(Self.pInstanceList);
    SetLength(Self.pInstanceList, Size+1);
    Self.pInstanceList[Size] := AObject;
  End;

  //---------------------------------------------------------------------------------

  Procedure TPkgMAnager.ClearInstanceList;
  Var i : Integer;
  Begin
    For i := 0 to Length(Self.pInstanceList)-1 do
      If Assigned(Self.pInstanceList[i]) Then
        FreeAndNil(Self.pInstanceList[i]);
  End;

  //---------------------------------------------------------------------------------

  Procedure TPkgManager.RemoveInstance(Index : Integer);
  Var Size : Integer;
  Begin
    Size := Length(Self.pInstanceList);
    If Index = Size-1 Then
      SetLength(Self.pInstanceList, Size-1)
    Else
    Begin
      Self.pInstanceList[index] := Self.pInstanceList[Size-1];
      SetLength(Self.pInstanceList, Size-1);
    End;
  End;

  //---------------------------------------------------------------------------------

  Procedure TPkgManager.UpdateInstanceList;
  Var i : Integer;
  Begin
    i := 0;
    While i < Length(Self.pInstanceList) do
    Begin
      If Not Assigned(Self.pInstanceList[i]) then
        Self.RemoveInstance(i);
      Inc(i);
    End;
  End;

  //---------------------------------------------------------------------------------
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité pkgCommon - manager de paquets

```
End.
```

I-D-2 - Explications du code

I-D-2.1 - Chargement/Déchargement

On retrouve les méthodes classiques de Chargement/Déchargement du paquet (LoadPackage/UnloadPackage).

Le chargement des classes se fait au fur et à mesure de la création des instances des objets. Deux méthodes existent : création d'instance de type **TObject** avec constructeur simple et une autre de type **TControl** nécessitant un paramètre **Owner**. D'autres méthodes peuvent être implémentées pour retourner un objet de classe la plus proche possible de l'objet final afin d'éviter un maximum de transtypage.

I-D-2.2 - InstanceList

Les méthodes **AddInstance**, **RemoveInstance**, **ClearInstanceList** et **UpdateInstanceList** servent à manipuler la liste **InstanceList**.

Voici son rôle :

Dans le cas des composants, c'est leur propriétaire (contenu dans la propriété **Owner**) qui est responsable de leur destruction. Or dans le cas de la gestion dynamique de paquets, il est possible que le paquet soit déchargé avant que le propriétaire des objets soit détruit (détruisant en même temps ses fils), ce qui entraîne une erreur de violation d'accès. En effet, il persiste en mémoire des objets dont la définition des méthodes n'existe plus (puisque les définitions se trouvent dans le paquet).

Dans ce cas, il est nécessaire de détruire toutes les instances créées à partir de classes des paquets chargés dynamiquement. D'où le stockage de la référence dans une liste lors de la création d'un composant. Cette liste sera parcourue lors du déchargement du paquet pour libérer toutes les instances de ces objets.

I-E - Aller plus loin

Je conseille aux créateurs de composants devant être chargés dynamiquement depuis un paquet de créer 2 classes par composant.

La première classe est une classe générique qui décrit les méthodes accessibles depuis l'application et éventuellement un comportement de base. Les méthodes devant être implémentées différemment seront déclarées **virtual**, et si le cas se présente **abstract**.

La seconde implémentera le fonctionnement spécifique à chaque plug-in en surchargeant les méthodes **virtual** de la classe générique.

But : posséder une classe de base connue de l'exécutable afin de posséder un maximum d'informations sur la classe exportée.

Le paquet contiendra donc une version améliorée de ces classes qui implémentera les méthodes spécifiques.

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Voici un exemple :

Classe connue de l'exécutable

```
Type
  TOperationClass = Class of TOperation;

  TOperation = Class(TControl) // ou inférieur
  Public
    Function Execute(Val1, Val2 : Integer) : Integer; Abstract; Virtual;
  End;
```

Classe du premier paquet

```
Type
  TAddition = Class(TOperation)
  Public
    Function Execute(Val1, Val2 : Integer) : Integer; Override;
  End;

Function TAddition.Execute(Val1, Val2 : Integer) : Integer;
Begin
  Result := Val1 + Val2;
End;
```

Classe du second paquet

```
Type
  TMultiplication = Class(TOperation)
  Public
    Function Execute(Val1, Val2 : Integer) : Integer; Override;
  End;

Function TMultiplication.Execute(Val1, Val2 : Integer) : Integer;
Begin
  Result := Val1 * Val2;
End;
```

Ainsi la méthode **Execute** est connue depuis l'exécutable, ce qui n'était pas le cas dans un chargement classique a partir de **TPersistentClass**.

Par contre son exécution varie en fonction de la classe chargée.

I-F - Conclusion

J'espère que ce tutoriel vous aura été profitable et que vous l'avez aprécié.

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

II - Utilisation des interfaces

La deuxième solution consiste en l'utilisation des interfaces, qui y reprennent tout leur sens premier, à savoir la **capacité** d'un objet à exécuter telle ou telle commande (les méthodes).

Cette méthode est (beaucoup) plus complexe que la précédente, autant pour la mise en oeuvre que pour la maintenance. Elle permet cependant une communications aisées avec le plug-in, via les méthodes des interfaces.

Un autre avantage de cette méthode est que l'on peut utiliser plusieurs interfaces : une (obligatoire) qui déclare les méthodes indispensables à l'intégration du composant dans l'exécutable ; et d'autres (facultatives) qui déclarent les méthodes permettant des *extra* (genre gestion du redimensionnement du composant, gestion de commandes que l'exécutable se charge d'insérer dans son menu, ...).

II-A - Un exemple concret

Nous allons commencer par un exemple concret relativement simple. Nous allons créer un TPanel dans le module extérieur (la dll ou le paquet) que nous allons insérer dans la fenêtre principale.

- **PluginIntf**, unité commune qui déclare l'interface.
- **TestPlugin**, unité principale de la dll/du paquet
- **MainForm**, fiche principale de l'application

Nous allons étudier le code de ces trois unités.

Unité PluginIntf - déclaration de l'interface IPlugin

```
unit PluginIntf;

interface

uses
  ExtCtrls;

type
  IPlugin = interface
    ['{F1F7186D-76E3-4DBD-8707-408C792B1C82}']
    function GetComponent : TPanel;
    property Component : TPanel read GetComponent;
  end;

implementation

end.
```

Unité TestPlugin - test de plug-in

```
unit TestPlugin;

uses
  ExtCtrls, PluginIntf;

type
  TTestPlugin = class(TInterfacedObject, IPlugin)
  private
    FPanel : TPanel;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité TestPlugin - test de plug-in

```
    public
      constructor Create;
      destructor Destroy; override;
      function GetComponent : TPanel;
    end;

  function CreatePlugin : IPlugin;

  exports CreatePlugin;

  implementation

  function CreatePlugin : IPlugin;
  begin
    Result := TTestPlugin.Create;
  end;

  constructor TTestPlugin.Create;
  begin
    inherited;
    FPanel := TPanel.Create(nil);
    with FPanel do
    begin
      Font.Size := 24;
      Font.Color := clBlue;
      Caption := 'Panel créé dans le plug-in';
      Width := 500;
      Height := 300;
    end;
  end;

  destructor TTestPlugin.Destroy;
  begin
    FPanel.Free;
    inherited;
  end;

  function TTestPlugin.GetComponent : TPanel;
  begin
    Result := FPanel;
  end;

  end.
```

Unité MainForm - fiche principale de l'application

```
unit MainForm;

uses
  Windows, Forms, PluginIntf, ...;

resourcestring
  sUnexistingFile = 'Le fichier ''%s'' n''existe pas';
  sDLLCantBeLoaded = 'La DLL ne peut être chargée';
  sPackageCantBeLoaded = 'Le package ne peut être chargé';
  sUnexistingFunction = 'La fonction %s n''existe pas';

type
  TFormMain = class(TForm)
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Déclarations privées }
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité MainForm - fiche principale de l'application

```
      PackHandle : HModule;
      Plugin : IPlugin;
    public
      { Déclarations publiques }
    end;

    EPluginError = class(Exception);

  implementation

  procedure TFormMain.FormCreate(Sender: TObject);
  begin
    PackHandle := 0;
    Plugin := nil;
  end;

  procedure TFormMain.Button1Click(Sender: TObject);
  type
    TCreatePluginFunc = function : IPlugin; stdcall;
  var CreatePlugin : TCreatePluginFunc;
  begin
    if PackHandle <> 0 then exit;
    try
      if not FileExists('TestPlugin.bpl') then
        raise EPluginError.CreateFmt(sUnexistingFile, [FFileName]);
      PackHandle := LoadPackage('TestPlugin.bpl');
      if PackHandle = 0 then
        raise EPluginError.Create(sPackageCantBeLoaded);
      CreatePlugin := GetProcAddress(PackHandle, 'CreatePlugin');
      if CreatePlugin = nil then
        raise EPluginError.CreateFmt(sUnexistingFunction, ['CreatePlugin']);
      Plugin := CreatePlugin;

      with Plugin.Component do
      begin
        ParentWindow := Form.Handle;
        SetParent(Handle, Form.Handle);
        Left := 0;
        Top := 0;
        Visible := True;
      end;
    except
      FormDestroy(nil);
      raise;
    end;
  end;

  procedure TFormMain.FormDestroy(Sender: TObject);
  begin
    if Plugin <> nil then with Plugin.Component do
    begin
      ParentWindow := nil;
      SetParent(Handle, 0);
    end;
    Plugin := nil;
    if PackHandle <> 0 then
      UnloadPackage(FHandle);
    PackHandle := 0;
  end;

  end.
```

Voici donc un exemple très simple (cependant beaucoup plus complexe que la première méthode) qui affiche un panel créé dans le plug-in dans la fenêtre de l'application. Mais ceci ne nous est pas d'une grande utilité. Nous allons

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

maintenant voir comment réaliser des plug-in performants et souples avec lesquels l'application peut communiquer au moyen des méthodes de différentes interfaces.

II-B - Déclaration de l'interface

La première étape est la déclaration de l'interface (ou des interfaces) déclarant les méthodes qui seront utilisées depuis l'exécutable. L'unité s'en occupant sera la seule commune à l'exécutable et à la dll/au paquet.

II-B-1 - L'indispensable dans l'unité

Dans l'unité commune doit être déclarée, au minimum, une interface qui déclare une méthode **GetComponent** et sa propriété associée **Component** (**read** GetComponent). Vous pouvez l'appeler comme vous voulez. Cette méthode doit renvoyer un objet de type **TAncetre** (qui doit être descendant de TWinControl).

Remarquez la présence d'une ligne **['{GUID}']**. Elle indique le GUID (Globally Unique IDentifier) de l'interface. Elle est obligatoire. Vous ne pouvez pas recopier directement cette ligne. Pour l'insérer dans votre code Delphi, vous devez entrer **Ctrl+Shift+G**, qui produira un GUID *pratiquement* unique (on ne fait pas mieux dans le genre que ce raccourci).

```
unit PluginIntf;

interface

uses
  ...;

type
  IPlugin = interface
    ['{F1F7186D-76E3-4DBD-8707-408C792B1C82}']
    function GetComponent : TAncetre;
    property Component : TAncetre read GetComponent;
  end;

implementation

end.
```

II-B-2 - Méthodes additionnelles de l'interface

Nous avons vu qu'une interface au moins doit être déclarée et que celle-ci doit proposer une propriété et son *getter* (méthode permettant de récupérer la valeur de la propriété). Vous pouvez cependant ajouter d'autres méthodes, tant qu'elles sont **indispensables au bon fonctionnement de l'application**. Si vous devez ajouter des méthodes optionnelles, je vous recommande de créer une autre interface qui pourra être implémentée par les plug-in.

II-B-3 - Interfaces optionnelles

Vous pouvez réaliser des interfaces optionnelles, qui pourront être implémentées ou non dans les classes de plug-in. Leurs méthodes peuvent être par exemple des événements que le plug-in voudrait intercepter, ou des méthodes indiquant comment le composant peut être redimensionné, etc.

Nous allons étudier trois cas d'interfaces additionnelles :

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

- Ajout de commandes du plug-in dans le menu de la fenêtre de l'application
- Redimensionnement du composant
- Evénements de l'application que le plug-in veut intercepter

II-B-3.1 - Ajout de commandes du plug-in dans le menu de la fenêtre de l'application

Le premier exemple d'interface additionnelle permettra au plug-in d'insérer des commandes dans le menu de l'application principale. Il est possible de l'adapter pour pouvoir aussi les insérer dans la barre d'outils si besoin est.

Voyons d'abord le nouveau code.

```
unit PluginIntf;

interface

uses
  ...;

const
  // Types de commandes pour IPluginCommands
  ComType_BigMenu = 'BigMenu';
  ComType_MenuSeparator = 'MenuSeparator';
  ComType_Menu = 'Menu';

type
  // Type de call-back pour l'ajout de commandes
  TEnumCommandsProc = procedure(ComType, Caption, Hint : string;
                                Shortcut : TShortCut; Bitmap : TBitmap;
                                OnExecute : TNotifyEvent) of object;

  // Interface obligatoire
  IPlugin = interface
    ['{F1F7186D-76E3-4DBD-8707-408C792B1C82}']
    ...
  end;

  // Interface pour les commandes
  IPluginCommands = interface
    ['{5E46D466-0D18-4733-84F8-D7F60EF00C71}']
    procedure EnumCommands(EnumCommandsProc : TEnumCommandsProc);
  end;

  // Autres interfaces
  ...

implementation

end.
```

Examinons ce code pour en retirer les informations à retenir. L'interface **IPluginCommands** déclare une méthode **EnumCommands** qui prend en paramètre une méthode call-back à appeler pour chaque commande. Le type de ce call-back est **TEnumCommandsProc**. Voici l'explication de ses paramètres :

Nom	Type	Description
ComType	**string**	Indique le type de commande. Les valeurs possibles sont représentées

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

		par les constantes de type de commandes définies plus haut dans l'unité. Elles seront détaillées plus bas.
Caption	**string**	Indique le texte du menu. Ce paramètre n'est pas utilisé avec le type **ComType_MenuSeparator**.
Hint	**string**	Indique le *hint* du menu. Ce paramètre n'est pas utilisé avec le type **ComType_MenuSeparator**
Shortcut	TShortcut	Indique le raccourci clavier du menu. Ce paramètre n'est utilisé qu'avec le type **ComType_Menu**
Bitmap	TBitmap	Bitmap associé à l'élément de menu. Pour ne pas avoir de bitmap, passez la valeur **nil** pour ce paramètre. La couleur transparente est la couleur **clTeal**. Ce paramètre n'est utilisé qu'avec le type **ComType_Menu**
OnExecute	TNotifyEvent	Evénement associé à la sélection du menu. Ce paramètre n'est utilisé qu'avec le type **ComType_Menu**

Voici la description des valeurs pour **ComType** :

Nom de constante	Description
ComType_BigMenu	Nouveau menu principal (5) après ceux déjà créés. Utilse les paramètres Caption et Hint uniquement.
ComType_MenuSeparator	Nouveau séparateur sous le dernier menu principal créé. N'utilise aucun paramètre supplémentaire. Si aucun menu principal n'a encore été créé, cette commande est ignorée.
ComType_Menu	

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

	Nouveau menu sous le dernier menu principal créé. Utilise tous les paramètres. Si aucun menu principal n'a encore été créé, cette commande est ignorée.

Il ne restera plus au plug-in qu'à implémenter la méthode **EnumCommands** et y appeler la méthode **EnumCommand** pour chaque menu à ajouter. Mais nous étudierons ceci plus tard.

II-B-3.2 - Redimensionnement du composant

Le deuxième exemple que nous allons étudier est une interface qui permet au composant d'être redimensionné. En fait dans l'application d'exemple que nous allons réaliser, la fenêtre s'ajuste à la taille du composant.

Un plug-in qui n'implémente pas cette interface ne peut être redimensionné. Dès le moment où un plug-in implémente cette interface, la fiche est redimensionnable.

Voici le code de cette interface :

```
unit PluginIntf;

interface

uses
  ...;

type
  // Interface obligatoire
  IPlugin = interface
    ['{F1F7186D-76E3-4DBD-8707-408C792B1C82}']
    ...
  end;

  // Interface pour le redimensionnement
  IPluginSizeable = interface
    ['{CD829089-E50B-41E4-94DD-41108282A762}']
    function GetMinWidth : integer;
    function GetMaxWidth : integer;
    function GetMinHeight : integer;
    function GetMaxHeight : integer;
    function GetCanMaximize : boolean;

    property MinWidth : integer read GetMinWidth;
    property MaxWidth : integer read GetMaxWidth;
    property MinHeight : integer read GetMinHeight;
    property MaxHeight : integer read GetMaxHeight;
    property CanMaximize : boolean read GetCanMaximize;
  end;

  // Autres interfaces
  ...

implementation

end.
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Les quatre premières propriétés de cette interface fonctionnent comme la propriété **Constraints** des contrôles de Borland. La dernière spécifie si on peut maximiser la fenêtre parente.

II-A-3.3 - Evénements de l'application que le plug-in veut intercepter

Le dernier exemple illustre une interface semblable au fonctionnement des événements en Java. Des interfaces qui implémentent des méthodes qui servent d'événements.

Nous n'allons voir ici qu'un événement, **QueryUnload**, qui sera appelé lorsque le plug-in est sur le point d'être déchargé.

Voici le code de cette interface :

```
unit PluginIntf;

interface

uses
  ...;

type
  // Interface obligatoire
  IPlugin = interface
    ['{F1F7186D-76E3-4DBD-8707-408C792B1C82}']
    ...
  end;

  // Interface pour les événements
  IPluginEvents = interface
    ['{262BDFEE-8DC4-47F7-8A64-2FB728C32FCE}']
    procedure QueryUnload(var CanUnload : boolean);
  end;

  // Autres interfaces
  ...

implementation

end.
```

Je suppose que rien ne vous aura échappé ici après ce que nous avons déjà vu.

Vous n'aurez plus qu'à adapter et ajouter vos propres événements.

II-B-4 - Code complet de l'unité PluginIntf

Unité PluginIntf - interfaces pour les plug-in

```
unit PluginIntf;

interface

uses
  ...;

const
  // Types de commandes pour IPluginCommands
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité PluginIntf - interfaces pour les plug-in

```
    ComType_BigMenu = 'BigMenu';
    ComType_MenuSeparator = 'MenuSeparator';
    ComType_Menu = 'Menu';

  type
    // Type de call-back pour l'ajout de commandes
    TEnumCommandsProc = procedure(ComType, Caption, Hint : string;
                                  Shortcut : TShortCut; Bitmap : TBitmap;
                                  OnExecute : TNotifyEvent) of object;

    // Interface obligatoire
    IPlugin = interface
      ['{F1F7186D-76E3-4DBD-8707-408C792B1C82}']
      function GetComponent : TAncetre;
      property Component : TAncetre read GetComponent;
    end;

    // Interface pour les commandes
    IPluginCommands = interface
      ['{5E46D466-0D18-4733-84F8-D7F60EF00C71}']
      procedure EnumCommands(EnumCommandsProc : TEnumCommandsProc);
    end;

    // Interface pour le redimensionnement
    IPluginSizeable = interface
      ['{CD829089-E50B-41E4-94DD-41108282A762}']
      function GetMinWidth : integer;
      function GetMaxWidth : integer;
      function GetMinHeight : integer;
      function GetMaxHeight : integer;
      function GetCanMaximize : boolean;

      property MinWidth : integer read GetMinWidth;
      property MaxWidth : integer read GetMaxWidth;
      property MinHeight : integer read GetMinHeight;
      property MaxHeight : integer read GetMaxHeight;
      property CanMaximize : boolean read GetCanMaximize;
    end;

    // Interface pour les événements
    IPluginEvents = interface
      ['{262BDFEE-8DC4-47F7-8A64-2FB728C32FCE}']
      procedure QueryUnload(var CanUnload : boolean);
    end;

  implementation

  end.
```

II-C - Utilisation côté DLL/paquet

Après avoir déclaré les interfaces, nous allons réaliser un plug-in de test qui implémentera toutes les interfaces définies ci-avant.

Remarquez que la classe du plug-in elle-même n'a pas d'ancêtre requis. Nous allons donc la faire dériver de **TInterfacedObject**, qui est plus efficace si la classe doit implémenter des interfaces.

II-C-1 - Code complet de l'unité TestPlugin

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité TestPlugin - plug-in de test pour notre application d'exemple

```
unit TestPlugin;

interface

uses
  PluginIntf, ...;

type
  // Composant qui sera effectivement inséré dans la fiche
  TMonComposant = class(TAncetre);
    ...
  end;

  // Classe implémentant les interfaces
  TTestPlugin = class(TInterfacedObject, IPlugin, IPluginCommands, IPluginSizeable, IPluginEvents)
  private
    FComponent : TMonComposant;

    procedure TestExecute(Sender : TObject);
  public
    constructor Create;
    destructor Destroy; override;

    // Implémentation de IPlugin
    function GetComponent : TAncetre;

    // Implémentation de IPluginCommands
    procedure EnumCommands(EnumCommandsProc : TEnumCommandsProc);

    // Implémentation de IPluginSizeable
    function GetMinWidth : integer;
    function GetMaxWidth : integer;
    function GetMinHeight : integer;
    function GetMaxHeight : integer;
    function GetCanMaximize : boolean;

    // Implémentation de IPluginEvents
    procedure QueryUnload(var CanUnload : boolean);
  end;

function CreatePlugin : IPlugin; stdcall;

exports CreatePlugin;

implementation

function CreatePlugin : IPlugin;
begin
  Result := TTestPlugin.Create;
end;

constructor TTestPlugin.Create;
begin
  inherited;
  FComponent := TMonComposant.Create(nil);
end;

destructor TTestPlugin.Destroy;
begin
  FComponent.Free;
  inherited;
end;

procedure TTestPlugin.TestExecute(Sender : TObject);
begin
  ShowMessage('Test d''événement');
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité TestPlugin - plug-in de test pour notre application d'exemple

```
end;

function TTestPlugin.GetComponent : TAncetre;
begin
  Result := FComponent;
end;

procedure TTestPlugin.EnumCommands(EnumCommandsProc : TEnumCommandsProc);
begin
  EnumCommandsProc(ComType_BigMenu, 'Test', 'Test de menu principal', 0, nil, nil);
  EnumCommandsProc(ComType_Menu, 'Test', 'Test de menu', 0, nil, TestExecute);
end;

function TTestPlugin.GetMinWidth : integer;
begin
  Result := 100;
end;

function TTestPlugin.GetMaxWidth : integer;
begin
  Result := 0;
end;

function TTestPlugin.GetMinHeight : integer;
begin
  Result := 200;
end;

function TTestPlugin.GetMaxHeight : integer;
begin
  Result := 0;
end;

function TTestPlugin.GetCanMaximize : boolean;
begin
  Result := True;
end;

procedure TTestPlugin.QueryUnload(var CanUnload : boolean);
begin
  CanUnload := MessageBox('Voulez-vous vraiment décharger ce plug-in ?',
                        mtConfirmation, [mbYes, mbNo]) = mrYes;
end;

end.
```

II-C-2 - Explications du code

Il n'y a pas grand chose à dire sur ce code, mais on peut s'attarder sur deux choses.

On peut remarquer que l'on a choisi la classe **TInterfacedObject** comme ancêtre pour la classe de plug-in. Ceci afin de ne pas devoir implémenter les méthodes **QueryInterface**, **_AddRef** et **_Release** que nous impose l'interface **IInterface**, ancêtre implicite de tout autre interface.

Une autre chose à ne pas rater est la fonction exportée **CreatePlugin**. Cette fonction sera appelée par le programme principal afin de récupérer une instance de type **IPlugin**. Il la libérera simplement en affectant **nil** à la variable la contenant.

Il me semble que le reste est aisément compréhensible d'autant plus que le but de chaque méthode a été vu dans les sections précédentes.

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

II-D - Utilisation côté application

Il ne reste plus maintenant qu'à charger notre plug-in dans l'application.

II-D-1 - Classe TPlugin

Afin de clarifier le code utile de l'application (donc tout sauf le code qui gère les plug-in), nous allons rédiger une classe **TPlugin** qui se chargera de créer/utiliser/détruire les plug-in. Examinons tout d'abord le code de l'unité **PluginClass**.

II-D-1.1 - Code complet de l'unité PluginClass

Unité PluginClass - classe de gestion des plug-in

```
unit PluginClass;

interface

uses
  PluginIntf, ...;

// Si vous activez la défition suivante, les plug-in seront chargés en tant que
// DLL, et non que packages. Les plug-in de type DLL sont obsolètes, d'où le
// nom de cette définition.
{.$DEFINE OLD_STYLE_PLUGIN}

resourcestring
  sUnexistingFile = 'Le fichier ''%s'' n''existe pas';
  sDLLCantBeLoaded = 'La DLL ne peut être chargée';
  sPackageCantBeLoaded = 'Le package ne peut être chargé';
  sUnexistingFunction = 'La fonction %s n''existe pas';

type
  EPluginError = class(Exception);

  TPlugin = class
  private
    FFileName : TFileName;

    FHandle : HModule;
    FPlugin : IPlugin;
    FPluginSupportsCommands : boolean;
    FPluginCommands : IPluginCommands;
    FPluginSupportsSizeable : boolean;
    FPluginSizeable : IPluginSizeable;
    FPluginSupportsEvents : boolean;
    FPluginEvents : IPluginEvents;

    FComponent : TAncetre;

    FImages : TImageList;
    FImagesCount : integer;
    FBigMenu : TMenuItem;
    FBigMenuCount : integer;
    FCommandsObjects : TObjectList;
    FCurrentBigMenu : TMenuItem;

    FMinWidth : integer;
    FMaxWidth : integer;
    FMinHeight : integer;
    FMaxHeight : integer;
    FCanMaximize : boolean;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité PluginClass - classe de gestion des plug-in

```
    function GetWidth : integer;
    function GetHeight : integer;
    procedure SetWidth(New : integer);
    procedure SetHeight(New : integer);

    procedure CreateInstance;
    procedure ReleaseInstance;
    procedure AddCommand(ComType, Caption, Hint : string;
                         ShortCut : TShortCut; Bitmap : Graphics.TBitmap;
                         OnExecute : TNotifyEvent);
    procedure AddCommands(Images : TImageList; BigMenu : TMenuItem);
    procedure RemoveCommands;
  public
    constructor Create(AFileName : TFileName);
    destructor Destroy; override;

    procedure Load(Form : TForm; Images : TImageList; BigMenu : TMenuItem;
                   TopOfComponent : integer);
    procedure UnLoad;

    property Width : integer read GetWidth write SetWidth;
    property Height : integer read GetHeight write SetHeight;
    property CanResize : boolean read FPluginSupportsSizeable;
    property MinWidth : integer read FMinWidth;
    property MaxWidth : integer read FMaxWidth;
    property MinHeight : integer read FMinHeight;
    property MaxHeight : integer read FMaxHeight;
    property CanMaximize : boolean read FCanMaximize;
  end;

implementation

constructor TPlugin.Create(AFileName : TFileName);
begin
  inherited Create;
  FFileName := AFileName;

  FHandle := 0;
  FPlugin := nil;
  FPluginSupportsCommands := False;
  FPluginCommands := nil;
  FPluginSupportsSizeable := False;
  FPluginSizeable := nil;
  FPluginSupportsEvents := False;
  FPluginEvents := nil;

  FComponent := nil;

  FImages := nil;
  FImagesCount := 0;
  FBigMenu := nil;
  FBigMenuCount := 0;
  FCommandsObjects := nil;
  FCurrentBigMenu := nil;

  FMinWidth := 0;
  FMaxWidth := 0;
  FMinHeight := 0;
  FMaxHeight := 0;
  FCanMaximize := False;
end;

destructor TPlugin.Destroy;
begin
  if FHandle <> 0 then ReleaseInstance;
  inherited Destroy;
end;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité PluginClass - classe de gestion des plug-in

```
function TPlugin.GetWidth : integer;
begin
  Result := FComponent.Width;
end;

function TPlugin.GetHeight : integer;
begin
  Result := FComponent.Height;
end;

procedure TPlugin.SetWidth(New : integer);
begin
  if CanResize and
     ((MinWidth = 0) or (New >= MinWidth)) and
     ((MaxWidth = 0) or (New <= MaxWidth)) then
    FComponent.Width := New;
end;

procedure TPlugin.SetHeight(New : integer);
begin
  if CanResize and
     ((MinHeight = 0) or (New >= MinHeight)) and
     ((MaxHeight = 0) or (New <= MaxHeight)) then
    FComponent.Height := New;
end;

procedure TPlugin.CreateInstance;
type
  TCreatePluginFunc = function : IPlugin; stdcall;
var CreatePlugin : TCreatePluginFunc;
begin
  if FHandle <> 0 then exit;
  try
    if not FileExists(FFileName) then
      raise EPluginError.CreateFmt(sUnexistingFile, [FFileName]);
    {$IFDEF OLD_STYLE_PLUGIN}
      FHandle := LoadLibrary(PChar(FFileName));
      if FHandle = 0 then
        raise EPluginError.Create(sDLLCantBeLoaded);
    {$ELSE}
      FHandle := LoadPackage(FFileName);
      if FHandle = 0 then
        raise EPluginError.Create(sPackageCantBeLoaded);
    {$ENDIF}
    CreatePlugin := GetProcAddress(FHandle, 'CreatePlugin');
    if CreatePlugin = nil then
      raise EPluginError.CreateFmt(sUnexistingFunction, ['CreatePlugin']);
    FPlugin := CreatePlugin;
    try
      FPluginCommands := FPlugin as IPluginCommands;
      FPluginSupportsCommands := True;
    except
    end;
    try
      FPluginSizeable := FPlugin as IPluginSizeable;
      FPluginSupportsSizeable := True;
      FMinWidth := FToolSizeable.MinWidth;
      FMaxWidth := FToolSizeable.MaxWidth;
      FMinHeight := FToolSizeable.MinHeight;
      FMaxHeight := FToolSizeable.MaxHeight;
      FCanMaximize := FToolSizeable.CanMaximize;
    except
    end;
    try
      FPluginEvents := FPlugin as IPluginEvents;
      FPluginSupportsEvents := True;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité PluginClass - classe de gestion des plug-in

```
      except
      end;
    except
      ReleaseInstance;
      raise;
    end;
  end;

  procedure TPlugin.ReleaseInstance;
  begin
    FMinWidth := 0;
    FMaxWidth := 0;
    FMinHeight := 0;
    FMaxHeight := 0;
    FCanMaximize := False;
    FPluginEvents := nil;
    FPluginSupportsEvents := False;
    FPluginSizeable := nil;
    FPluginSupportsSizeable := False;
    FPluginCommands := nil;
    FPluginSupportsCommands := False;
    FPlugin := nil;
    if FHandle <> 0 then
      {$IFDEF OLD_STYLE_PLUGIN}
        FreeLibrary(FHandle);
      {$ELSE}
        UnloadPackage(FHandle);
      {$ENDIF}
    FHandle := 0;
  end;

  procedure TPlugin.AddCommand(ComType, Caption, Hint : string;
                               Shortcut : TShortCut; Bitmap : Graphics.TBitmap;
                               OnExecute : TNotifyEvent);
  var Action : TAction;
      MenuItem : TMenuItem;
      Bmp : Graphics.TBitmap;
      I : integer;
  begin
    if ComType = ComType_BigMenu then
    begin
      FCurrentBigMenu := TMenuItem.Create(nil);
      FCurrentBigMenu.Caption := Caption;
      FCurrentBigMenu.Hint := Hint;
      FBigMenu.Add(FCurrentBigMenu);
      FCommandsObjects.Add(FCurrentBigMenu);
    end else
    if ComType = ComType_MenuSeparator then
    begin
      if FCurrentBigMenu = nil then exit;
      MenuItem := TMenuItem.Create(nil);
      MenuItem.Caption := '-';
      FCurrentBigMenu.Add(MenuItem);
      FCommandsObjects.Add(MenuItem);
    end else
    if ComType = ComType_Menu then
    begin
      if FCurrentBigMenu = nil then exit;
      Action := TAction.Create(nil);
      Action.Caption := Caption;
      Action.Hint := Hint;
      Action.ShortCut := ShortCut;
      if (Bitmap <> nil) and (not Bitmap.Empty) then
      begin
        Bmp := Graphics.TBitmap.Create;
        Bmp.Width := 16;
        Bmp.Height := 16;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité PluginClass - classe de gestion des plug-in

```
      Bmp.Canvas.CopyRect(Rect(0, 0, 16, 16),
                          Bitmap.Canvas,
                          Rect(0, 0, Bitmap.Width, Bitmap.Height));
      FImages.AddMasked(Bmp, clTeal);
      Bmp.Free;
      Action.ImageIndex := FImages.Count-1;
    end;
    Action.OnExecute := OnExecute;
    FCommandsObjects.Add(Action);
    MenuItem := TMenuItem.Create(nil);
    MenuItem.Action := Action;
    FCurrentBigMenu.Add(MenuItem);
    FCommandsObjects.Add(MenuItem);
  end;
end;

procedure TPlugin.AddCommands(Images : TImageList; BigMenu : TMenuItem);
begin
  if not FPluginSupportsCommands then exit;

  FImages := Images;
  FImagesCount := FImages.Count;
  FBigMenu := BigMenu;
  FBigMenuCount := FBigMenu.Count;
  FCommandsObjects := TObjectList.Create;

  FCurrentBigMenu := nil;
  FPluginCommands.EnumCommands(AddCommand);
end;

procedure TPlugin.RemoveCommands;
begin
  if not FPluginSupportsCommands then exit;

  while FBigMenu.Count > FBigMenuCount do
    FBigMenu.Delete(FBigMenuCount);
  while FImages.Count > FImagesCount do
    FImages.Delete(FImagesCount);
  FImages := nil;
  FImagesCount := 0;
  FBigMenu := nil;
  FBigMenuCount := 0;
  FCommandsObjects.Free;
end;

procedure TPlugin.Load(Form : TForm; Images : TImageList; BigMenu : TMenuItem;
                       TopOfComponent : integer);
begin
  try
    CreateInstance;
    FComponent := FPlugin.Component;
    with FComponent do
    begin
      ParentWindow := Form.Handle;
      SetParent(Handle, Form.Handle);
      Left := 0;
      Top := TopOfComponent;
      Visible := True;
    end;

    AddCommands(Images, BigMenu);
  except
    on Error : Exception do
      raise EPluginError.Create(Error.Message);
  end;
end;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité PluginClass - classe de gestion des plug-in

```
procedure TPlugin.Unload;
begin
  if FHandle = 0 then exit;
  RemoveCommands;
  FComponent.ParentWindow := 0;
  SetParent(FComponent.Handle, 0);
  FComponent := nil;
  ReleaseInstance;
end;

end.
```

Ouf ! Ca en valait la peine hein ? Mais grâce à cette classe nous allons pouvoir gérer nos plug-in très simplement. La seule chose qui restera à gérer est le redimensionnement de la fenêtre.

II-D-1.2 - Etude du code

Observons d'un peu plus près quelques parties de ce code.

II-D-1.2a - Méthode CreateInstance

Méthode CreateInstance

```
procedure TPlugin.CreateInstance;
type
  TCreatePluginFunc = function : IPlugin; stdcall;
var CreatePlugin : TCreatePluginFunc;
begin
  if FHandle <> 0 then exit;
  try
    if not FileExists(FFileName) then
      raise EPluginError.CreateFmt(sUnexistingFile, [FFileName]);
    {$IFDEF OLD_STYLE_PLUGIN}
      FHandle := LoadLibrary(PChar(FFileName));
      if FHandle = 0 then
        raise EPluginError.Create(sDLLCantBeLoaded);
    {$ELSE}
      FHandle := LoadPackage(FFileName);
      if FHandle = 0 then
        raise EPluginError.Create(sPackageCantBeLoaded);
    {$ENDIF}
    CreatePlugin := GetProcAddress(FHandle, 'CreatePlugin');
    if CreatePlugin = nil then
      raise EPluginError.CreateFmt(sUnexistingFunction, ['CreatePlugin']);
    FPlugin := CreatePlugin;
    try
      FPluginCommands := FPlugin as IPluginCommands;
      FPluginSupportsCommands := True;
    except
    end;
    try
      FPluginSizeable := FPlugin as IPluginSizeable;
      FPluginSupportsSizeable := True;
      FMinWidth := FToolSizeable.MinWidth;
      FMaxWidth := FToolSizeable.MaxWidth;
      FMinHeight := FToolSizeable.MinHeight;
      FMaxHeight := FToolSizeable.MaxHeight;
      FCanMaximize := FToolSizeable.CanMaximize;
    except
    end;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Méthode CreateInstance

```
      try
        FPluginEvents := FPlugin as IPluginEvents;
        FPluginSupportsEvents := True;
      except
      end;
    except
      ReleaseInstance;
      raise;
    end;
  end;
```

Le centre de cette unité, c'est la méthode **CreateInstance**. Cette méthode charge la dll/le paquet et crée une instance du plug-in. Elle vérifie ensuite les interfaces supportées par la classe de plug-in. Nous utilisons pour cela l'**interrogation d'interface**, d'où les GUID indispensables.

On commence par vérifier que la dll/le paquet n'est pas déjà chargé(e). Ensuite, on le charge avec une compilation conditionnelle selon que l'on utilise des plug-in de type DLL ou Package.

Ensuite on récupère l'adresse de la fonction exportée **CreatePlugin**. On crée via cette fonction une nouvelle instance de la classe de plug-in, récupérée sous forme d'interface **IPlugin**.

Finalement, on teste l'implémentation des interfaces facultatives et on récupère, le cas échéant, des références à l'objet sous les formes des autres interfaces, c'est-à-dire la même adresse mais de type différent. Ceci afin de pouvoir utiliser les méthode de chaque interface plus facilement.

II-D-1.2b - Méthode ReleaseInstance

Méthode ReleaseInstance

```
procedure TPlugin.ReleaseInstance;
begin
  FMinWidth := 0;
  FMaxWidth := 0;
  FMinHeight := 0;
  FMaxHeight := 0;
  FCanMaximize := False;
  FPluginEvents := nil;
  FPluginSupportsEvents := False;
  FPluginSizeable := nil;
  FPluginSupportsSizeable := False;
  FPluginCommands := nil;
  FPluginSupportsCommands := False;
  FPlugin := nil;
  if FHandle <> 0 then
    {$IFDEF OLD_STYLE_PLUGIN}
      FreeLibrary(FHandle);
    {$ELSE}
      UnloadPackage(FHandle);
    {$ENDIF}
  FHandle := 0;
end;
```

Cette méthode sert à libérer les objets/ressources créées avec la méthode **CreateInstance**. Remarquez qu'il suffit d'affecter **nil** à toutes les variables référençant l'interface pour que celle-ci soit libérée.

Ici aussi le module est libéré différemment selon que c'est une DLL ou un paquet.

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

II-D-1.2c - Méthodes AddCommand et AddCommands

Méthodes AddCommand et AddCommands

```
procedure TPlugin.AddCommand(ComType, Caption, Hint : string;
                             Shortcut : TShortCut; Bitmap : Graphics.TBitmap;
                             OnExecute : TNotifyEvent);
var Action : TAction;
    MenuItem : TMenuItem;
    Bmp : Graphics.TBitmap;
    I : integer;
begin
  if ComType = ComType_BigMenu then
  begin
    FCurrentBigMenu := TMenuItem.Create(nil);
    FCurrentBigMenu.Caption := Caption;
    FCurrentBigMenu.Hint := Hint;
    FBigMenu.Add(FCurrentBigMenu);
    FCommandsObjects.Add(FCurrentBigMenu);
  end else
  if ComType = ComType_MenuSeparator then
  begin
    if FCurrentBigMenu = nil then exit;
    MenuItem := TMenuItem.Create(nil);
    MenuItem.Caption := '-';
    FCurrentBigMenu.Add(MenuItem);
    FCommandsObjects.Add(MenuItem);
  end else
  if ComType = ComType_Menu then
  begin
    if FCurrentBigMenu = nil then exit;
    Action := TAction.Create(nil);
    Action.Caption := Caption;
    Action.Hint := Hint;
    Action.ShortCut := ShortCut;
    if (Bitmap <> nil) and (not Bitmap.Empty) then
    begin
      Bmp := Graphics.TBitmap.Create;
      Bmp.Width := 16;
      Bmp.Height := 16;
      Bmp.Canvas.CopyRect(Rect(0, 0, 16, 16),
                          Bitmap.Canvas,
                          Rect(0, 0, Bitmap.Width, Bitmap.Height));
      FImages.AddMasked(Bmp, clTeal);
      Bmp.Free;
      Action.ImageIndex := FImages.Count-1;
    end;
    Action.OnExecute := OnExecute;
    FCommandsObjects.Add(Action);
    MenuItem := TMenuItem.Create(nil);
    MenuItem.Action := Action;
    FCurrentBigMenu.Add(MenuItem);
    FCommandsObjects.Add(MenuItem);
  end;
end;

procedure TPlugin.AddCommands(Images : TImageList; BigMenu : TMenuItem);
begin
  if not FPluginSupportsCommands then exit;

  FImages := Images;
  FImagesCount := FImages.Count;
  FBigMenu := BigMenu;
  FBigMenuCount := FBigMenu.Count;
  FCommandsObjects := TObjectList.Create;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Méthodes AddCommand et AddCommands

```
    FCurrentBigMenu := nil;
    FPluginCommands.EnumCommands(AddCommand);
  end;
```

Ces deux méthodes s'occupent de la création des éléments de menus avec la méthode **EnumCommands** de l'interface **IPluginCommands**, si le plug-in implémente celle-ci.

AddCommand est la méthode de call-back de la méthode **EnumCommands**, elle-même appelée par **AddCommands**.

II-D-1.2d - Méthodes Load et Unload

Méthodes Load et Unload

```
procedure TPlugin.Load(Form : TForm; Images : TImageList; BigMenu : TMenuItem;
                       TopOfComponent : integer);
begin
  try
    CreateInstance;
    FComponent := FPlugin.Component;
    with FComponent do
    begin
      ParentWindow := Form.Handle;
      SetParent(Handle, Form.Handle);
      Left := 0;
      Top := TopOfComponent;
      Visible := True;
    end;

    AddCommands(Images, BigMenu);
  except
    on Error : Exception do
      raise EPluginError.Create(Error.Message);
  end;
end;

procedure TPlugin.Unload;
begin
  if FHandle = 0 then exit;
  RemoveCommands;
  FComponent.ParentWindow := 0;
  SetParent(FComponent.Handle, 0);
  FComponent := nil;
  ReleaseInstance;
end;
```

La méthode **Load** charge effectivement le plug-in. Elle appelle d'abord la méthode **CreateInstance**, puis elle insère le composant dans la fiche passée en paramètre avant d'ajouter les commandes du plug-in, si le cas se présente.

La méthode **Unload** fait le contraire : elle supprime les commandes, retire le composant de la fiche et détruit l'instance via **ReleaseInstance**.

II-D-2 - Le code de la fenêtre principale

Nous considérerons ici que la fenêtre comporte un menu, une barre d'outils et une barre de statut, ainsi que le composant du plug-in une fois celui-ci chargé. Celle-ci ne devra plus s'occuper de charger/décharger le plug-in

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

puisqu'elle utilisera une instance de la classe **TPlugin**, définie ci-avant. Elle devra cependant s'occuper du redimensionnement du composant.

Unité MainForm - fiche principale de l'application

```
unit MainForm;

interface

uses
  PluginClass, ...;

type
  TFormMain = class(TForm)
    ImageList: TImageList;
    ActionList: TActionList;
    ActionCharger: TAction;
    ActionDecharger: TAction;
    BigMenu: TMainMenu;
    BigMenuFichier: TMenuItem;
    MenuCharger: TMenuItem;
    MenuDecharger: TMenuItem;
    ToolBar: TToolBar;
    ToolButtonCharger: TToolButton;
    ToolButtonDecharger: TToolButton;
    StatusBar: TStatusBar;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure ActionChargerExecute(Sender: TObject);
    procedure ActionDechargerExecute(Sender: TObject);
    procedure ResizePlugin(Sender: TObject);
  private
    { Déclarations privées }
    Plugin : TPlugin;
  public
    { Déclarations publiques }
  end;

var
  FormMain: TFormMain;

implementation

procedure TFormMain.FormCreate(Sender: TObject);
begin
  Plugin := TPlugin.Create('TestPlugin.bpl');
end;

procedure TFormMain.FormDestroy(Sender: TObject);
begin
  Plugin.Free;
end;

procedure TFormMain.ActionChargerExecute(Sender: TObject);
begin
  Plugin.Load(Self, BigMenu.Items, ImageList, ToolBar.Height);
  ClientWidth := Plugin.Width;
  ClientHeight := Plugin.Height+ToolBar.Height+StatusBar.Height;
  if Plugin.CanResize then
  begin
    BorderStyle := bsSizeable;
    with Constraints do
    begin
      MinWidth := Plugin.MinWidth;
      MaxWidth := Plugin.MaxWidth;
      MinHeight := Plugin.MinHeight+ToolBar.Height+StatusBar.Height;
      MaxHeight := Plugin.MaxHeight+ToolBar.Height+StatusBar.Height;
```


Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Unité MainForm - fiche principale de l'application
```
      end;
      if Plugin.CanMaximize then
        BorderIcons := [biSystemMenu, biMinimize, biMaximize];
      OnResize := ResizePlugin;
    end;
  end;

  procedure TFormMain.ActionDechargerExecute(Sender: TObject);
  begin
    OnResize := nil;
    Plugin.Unload;
    BorderIcons := [biSystemMenu, biMinimize];
    width Constraints do
    begin
      MinWidth := 0;
      MaxWidth := 0;
      MinHeight := 0;
      MaxHeight := 0;
    end;
    BorderStyle := bsSingle;
    ClientWidth := 300;
    ClientHeight := 200;
  end;

  procedure TFormMain.ResizePlugin(Sender: TObject);
  begin
    Plugin.Width := ClientWidth;
    Plugin.Height := ClientHeight-ToolBar.Height-StatusBar.Height;
  end;

  end.
```

C'est pas plus compliqué que ça. Et ça le serait encore moins si on n'avait pas accepter le redimensionnement des plug-in.

Je suppose qu'il n'est pas besoin de décortiquer ce petit bout de code, que vous aurez compris au premier coup d'oeil, surtout si vous avez bien compris ce que fait la classe **TPlugin**.

II-E - Conclusion

Cette méthode est enfin terminée. J'espère que vous aurez compris son fonctionnement et qu'elle vous sera profitable.

Vous pouvez **télécharger un exemple de programme avec les sources** (légèrement modifiées pour les besoins du MDI) et deux paquets de plug-in d'exemple.

Si ce lien ne fonctionne pas chez vous, utilisez **celui-ci**.

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Remerciements

Merci à **Nono40** et **Laurent Dardenne** pour leurs relectures et corrections d'orthographe et de forme.

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

1 : Package : terme anglais pour paquet
2 : Déchargement : on parle de déchargement pour un package comme ou parle de libération pour une DLL
3 : BPL : paquet en mode exécution
4 : DCP : paquet en mode conception
5 : Menu principal : menu d'en-tête tel que Fichier, Edition, ...

Copyright © 2005 Developpez LLC. Tout droits réservés Developpez LLC. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents et images sans l'autorisation expresse de Developpez LLC. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.