Partie II : Améliorer un composant par héritage

par Sébastien Doeraene

Date de publication : 09/08/2005

Dernière mise à jour :

Cette seconde partie va vous apprendre à améliorer un composant existant en héritant de celui-ci. En particulier, vous améliorerez le composant TImage pour lui ajouter une fonctionnalité de glisser-déposer.

Ce tutoriel a été écrit avec le support de Delphi 2005 édition Architecte. Si vous possédez une version antérieure ou une édition inférieure, il est possible que vous ne puissiez pas utiliser certaines des fonctionnalités présentées ici.

Certains liens dans ce tutoriel font référence à l'aide locale de Delphi 2005 - ils commencent tous par "ms-help://" -, ces liens ne fonctionnent que si vous avez enregistré ce tutoriel sur votre disque local et que vous possédez une version de Delphi supérieure ou égale à la version 2005.

Introduction
I - Choix du composant à développer
II - Fonctionnement du composant
III - Type de l'événement OnDrop
IV - Les différentes propriétés de TDropImage
V - Constructeur et destructeur
VI - Implémentation du glisser-déposer
VII - Sécuriser la destruction du contrôle DropControl
VIII - Code complet du composant TDropImage
IX - Test du composant
X - Intégration dans la palette des composants
XI - Conclusion
XII - Liens utiles
XIII - Remerciements

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Introduction

Maintenant que nous savons tout de la création de composants non-visuels, il est temps de s'attaquer aux composants visuels.

Nous allons donc commencer par voir comment il est possible d'améliorer des composants existants par la technique de l'héritage. Cette technique est certainement la plus importante, surtout au niveau didactique, puisqu'elle met en jeu un principe clef de la POO : la réutilisation du code.

```
type
  TLabelAmeliore = class(TLabel)
    ...
  end;
```

Comme nous l'avions fait dans la première partie, nous allons apprendre en pratiquant : nous allons créer un composant améliorant un composant existant.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

I - Choix du composant à développer

Pour cet exemple, nous allons développer un composant image que l'on peut *emporter* en cliquant dessus et en déplaçant la souris, puis en relâchant le bouton de la souris pour le *déposer*. On appelle cela le *drag 'n' drop* en anglais, littéralement *glisser-déposer*.

Une fois déposé sur la fiche, un **TImage** est figé : l'utilisateur n'a pas la possibilité de le déplacer. Nous allons donc améliorer ce composant pour permettre à l'utilisateur de *prendre* l'image et la déposer quelque part.

C'est un composant très utile par exemple si vous voulez réaliser un éditeur quelconque par glisser-déposer.

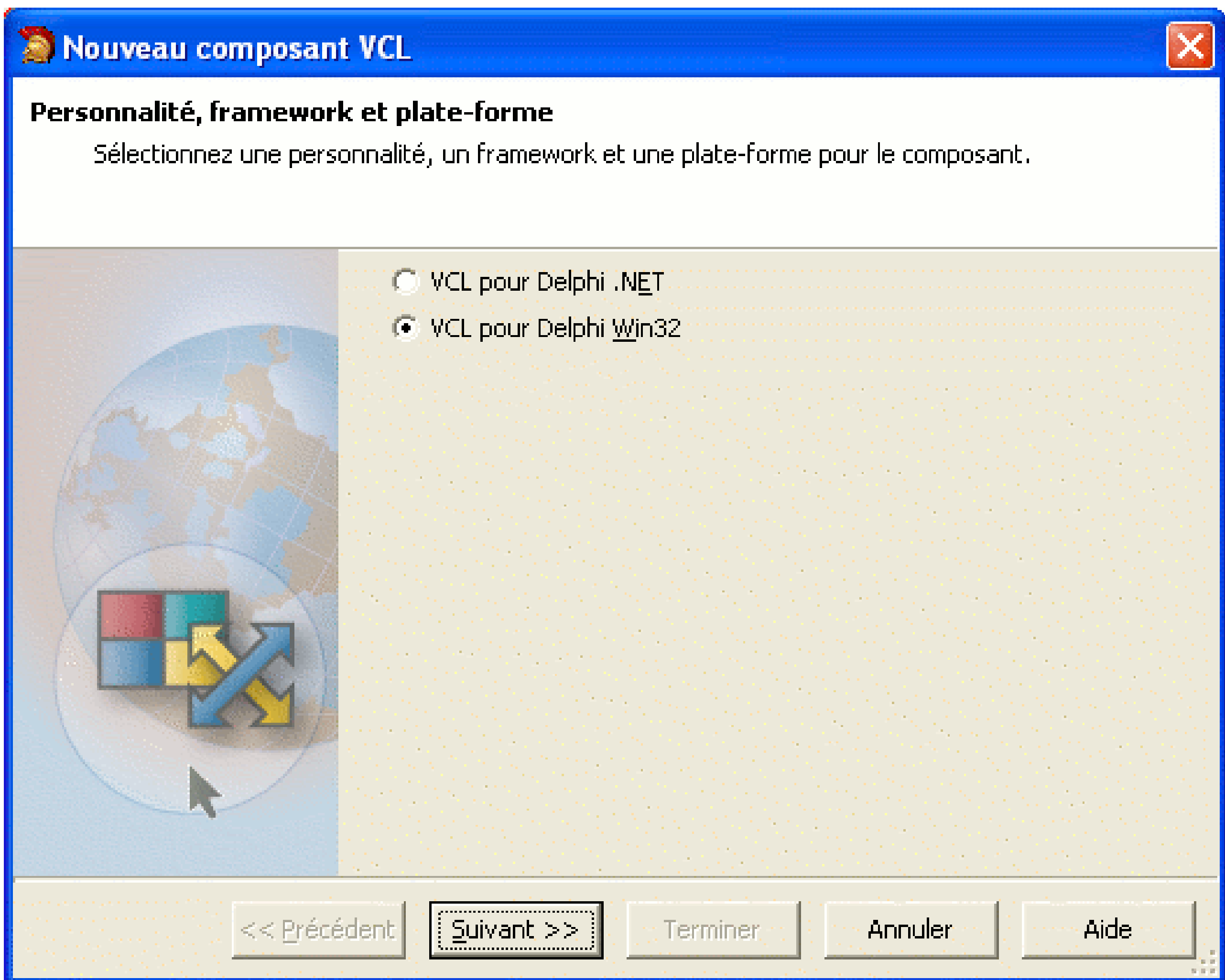
Nous nommerons ce composant **TDropImage** ("drop" signifie "laisser tomber" en anglais), et il héritera bien entendu de la classe **TImage**. Créez à cet effet une nouvelle unité dans le package **ComposTutoR** (voir la première partie, section II-B), que vous nommerez **DropImg.pas**.

```
unit DropImg.pas;

interface

uses
  ExtCtrls;

type
  TDropImage = class(TImage)
  end;

implementation

end.
```

Si vous préférez, vous pouvez également créer ce squelette *via* la commande **Composant|Nouveau Composant VCL...** (**Nouveau Composant...** pour Delphi 7 et antérieurs). Voici la démarche avec Delphi 2005.

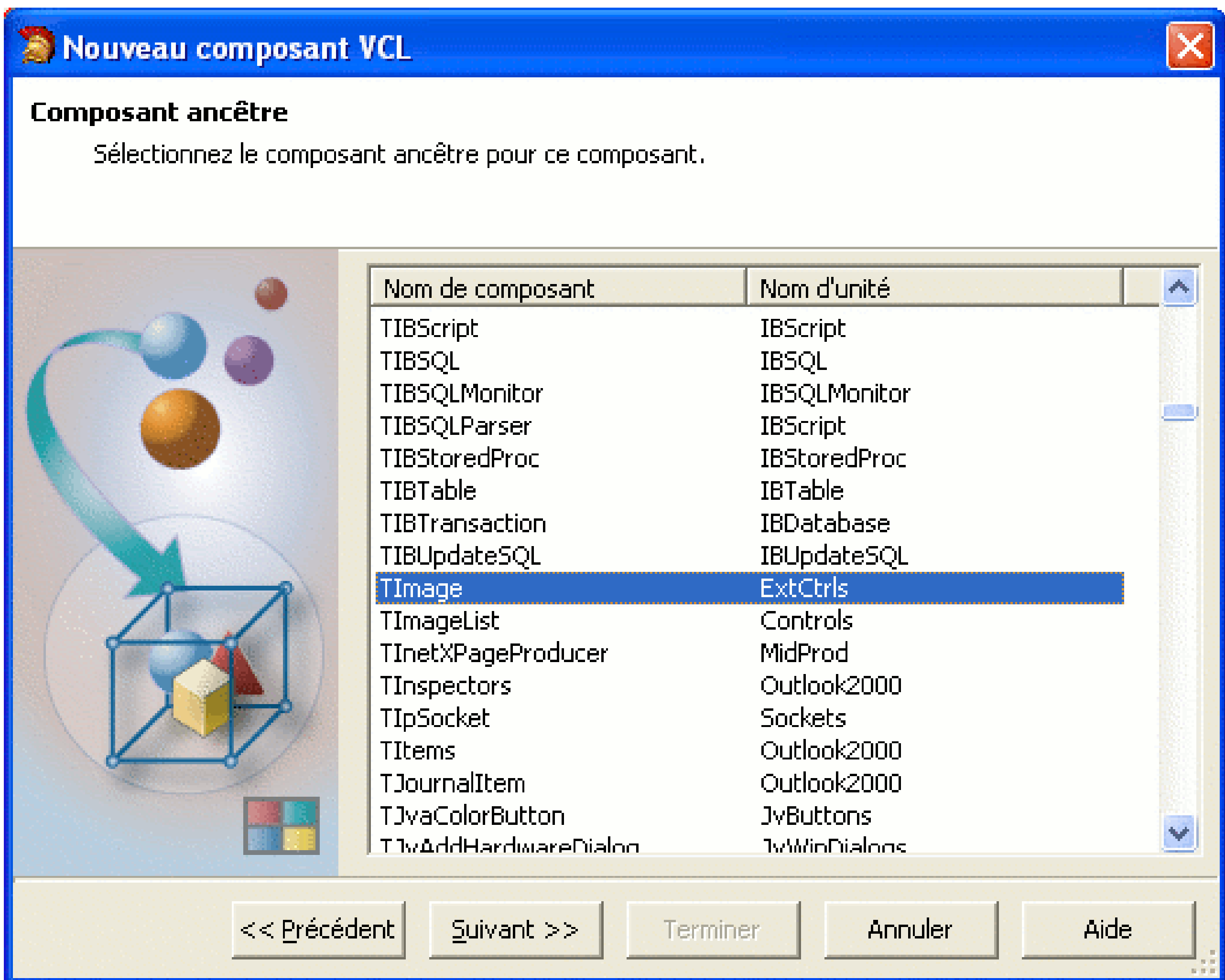
Après avoir sélectionné cet élément de menu, la boîte de dialogue suivante apparaît :

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.



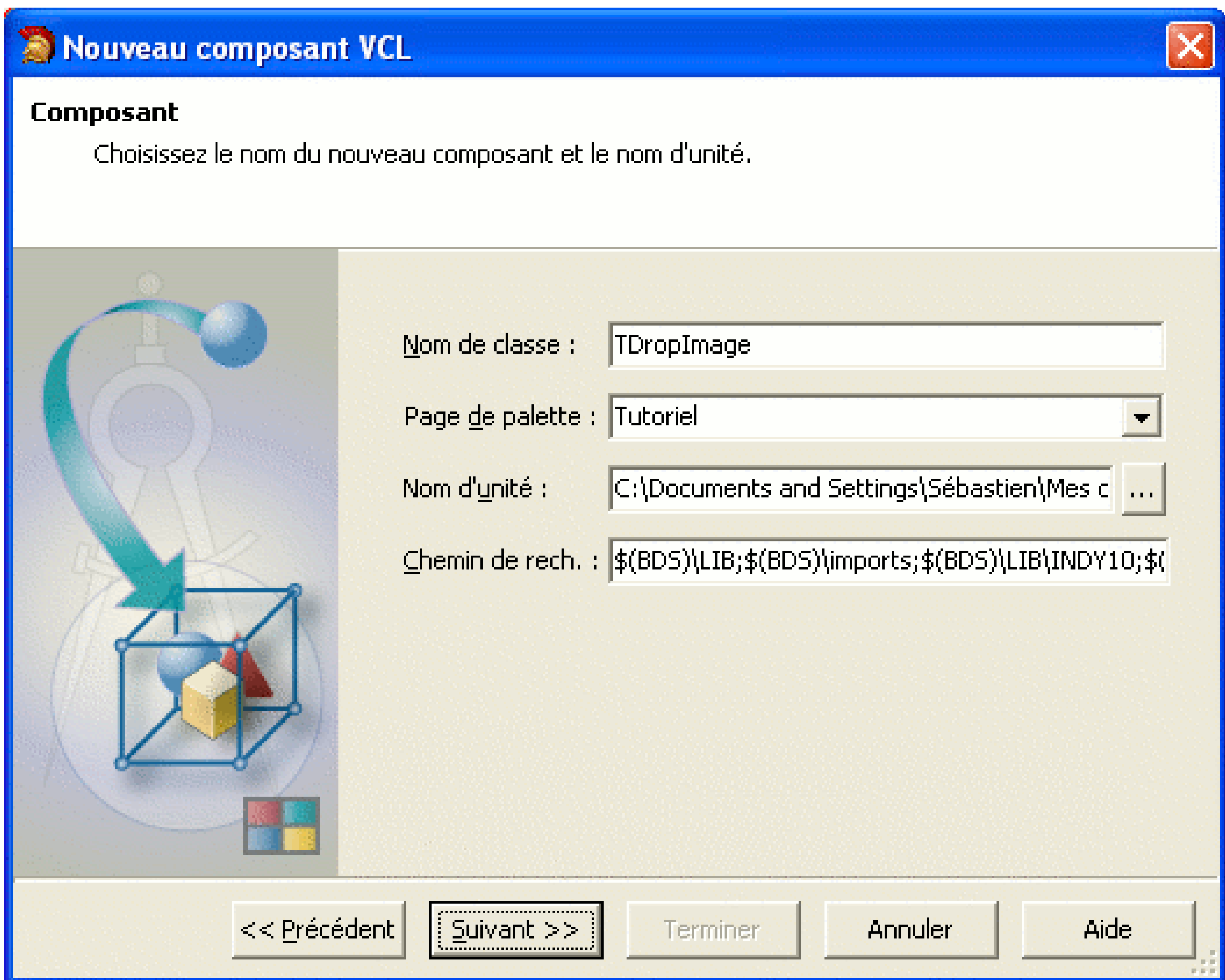
Il n'y a pas d'équivalent à cette page dans Delphi 7, puisque cela concerne .NET. Après avoir sélectionné **Win32** (par défaut), cliquez sur **Suivant**. La page suivante apparaît :

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://sjrd.developpez.com/delphi/tutoriel/composants/heritage/



Sur cette page, vous devez sélectionner le composant ancêtre de votre nouveau composant. Dans ce cas sélectionnez **TImage**, puis cliquez sur **Suivant**. La page suivante demande quelques informations supplémentaires sur le composant, notamment son nom de classe.

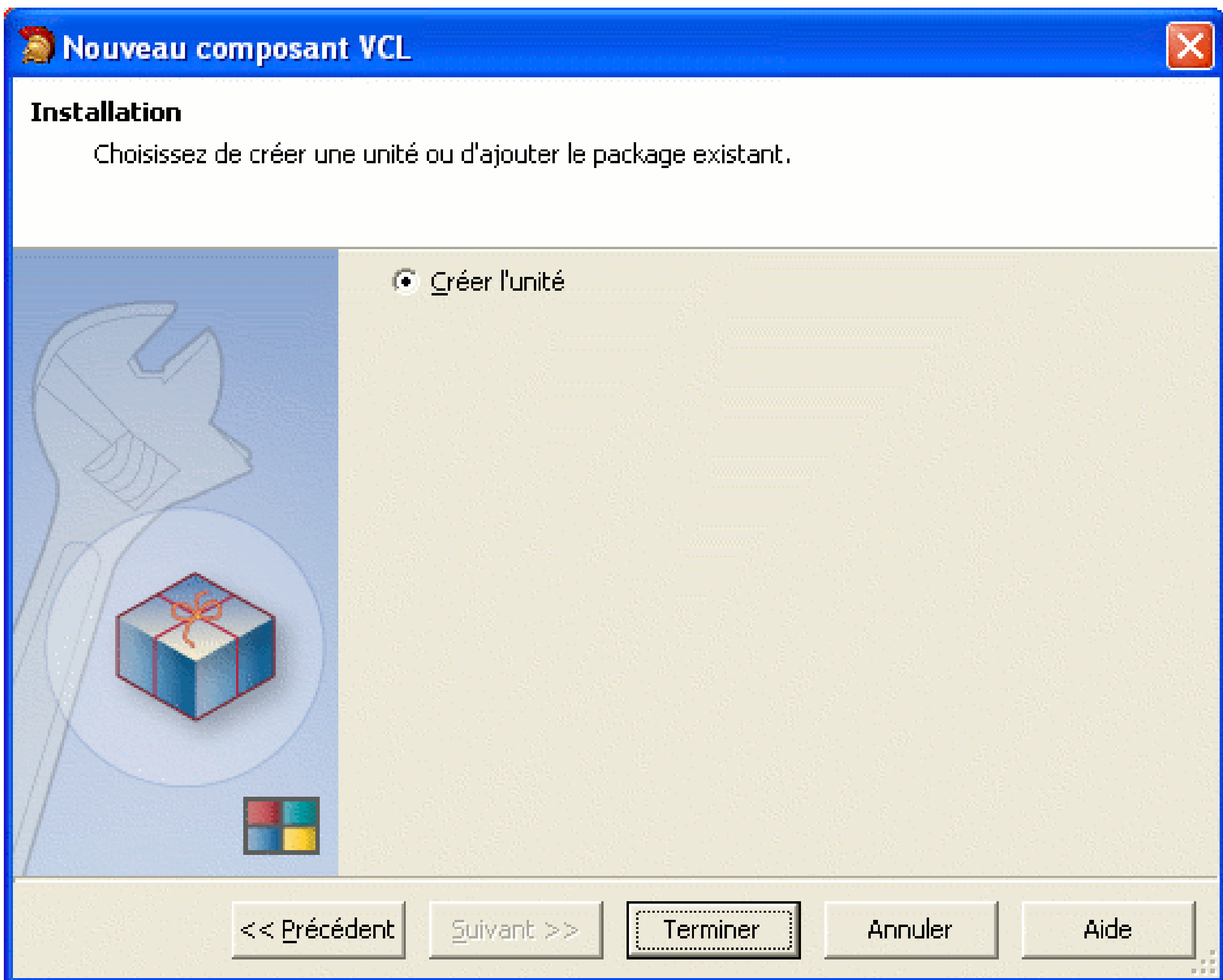
Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://sjrd.developpez.com/delphi/tutoriel/composants/heritage/



La page de la palette est la catégorie dans laquelle apparaîtra votre composant dans la palette des composants. Cette information sert ici à générer la procédure **Register**. Voilà ma raison de ne pas aimer cette démarche : cela génère cette procédure dans la même unité que le composant, alors qu'il faut la placer dans un autre package, comme nous l'avons fait jusqu'à présent.

La dernière page vous permet de générer l'unité :

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.



*Une attitude plus professionnelle aurait été de créer une classe **TCustomDropImage** dont les nouvelles propriétés seraient déclarées **protected**, puis d'en dériver une classe **TDropImage** où l'on redéfinirait ces propriétés comme **published**. Cela permettrait de créer un autre composant améliorant encore le **TDropImage** mais ne souhaitant pas que certaines de ses propriétés soient publiques. Nous laisserons ce point de côté pour des raisons de facilité.*

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

II - Fonctionnement du composant

Commençons par étudier comment va fonctionner notre composant dans une application.

Le principe en est assez simple : lorsqu'on enfonce le bouton gauche de la souris, on crée un *clone* de l'image qui va se déplacer au gré des mouvements de la souris. Une fois qu'on relâche le bouton gauche, on détruit le clone et on envoie un événement pour indiquer les coordonnées du *largage*.

Les coordonnées seront calculées de deux façons différentes, selon qu'un contrôle a été désigné ou non comme celui sur lequel doivent être déposées les images.

Dans le premier cas, on donnera les coordonnées par rapport à ce contrôle. Dans le deuxième, on donnera les coordonnées par rapport au parent de notre **TDropImage**.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://sjrd.developpez.com/delphi/tutoriel/composants/heritage/

III - Type de l'événement OnDrop

Bien entendu, le fait que l'utilisateur dépose l'image sera signalé par un événement. Cet événement se nommera **OnDrop** et sera du type **TDropEvent**.

Cet événement devra fournir en paramètre, en plus de **Sender**, les coordonnées où a été déposée l'image. Il se déclarera donc comme suit :

```
type
  TDropEvent = procedure(Sender : TObject; X, Y : integer) of object;
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

IV - Les différentes propriétés de TDropImage

En plus de l'événement **OnDrop**, le composant **TDropImage** devra posséder une propriété **DropControl** de type **TControl** indiquant sur quel contrôle l'image doit être déposée. Ce contrôle faisant normalement partie du parent du **TDropImage**.

De plus, la propriété **Enabled** déclarée dans **TImage** empêchera, si elle est placée à **False**, de prendre l'image pour la déposer.

Finalement, nous aurons besoin d'une variable privée de type **TImage** pour le clone.

Voici donc à quoi ressemble le code de notre classe pour l'instant :

```
TDropImage = class(TImage)
private
  FClone : TImage;

  FDropControl : TControl;
  FOnDrop : TDropEvent;
public
  constructor Create(AOwner : TComponent); override;
  destructor Destroy; override;
published
  property DropControl : TControl read FDropControl write FDropControl;
  property OnDrop : TDropEvent read FOnDrop write FOnDrop;
end;
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

V - Constructeur et destructeur

Les constructeur et destructeur sont très simples. Le destructeur se charge juste de libérer le clone si il est alloué.

```
constructor TDropImage.Create(AOwner : TComponent);
begin
  inherited;
  FClone := nil;
  FDropControl := nil;
  FOnDrop := nil;
end;

destructor TDropImage.Destroy;
begin
  if FClone <> nil then
    FClone.Free;
  inherited;
end;
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

VI - Implémentation du glisser-déposer

Pour implémenter le glisser-déposer, nous devrons intercepter trois événements : **OnMouseDown**, **OnMouseMove** et **OnMouseUp**.

Cependant, il n'est pas question ici d'utiliser les événements pour les intercepter, sinon un programme utilisant notre composant ne pourrait plus réagir à ces événements.

Heureusement, la VCL est bien faite et il existe des méthodes dynamiques protégées (**protected**) qui sont appelées lorsque la majorité des événements sont interceptés.

Nous allons donc surcharger les trois méthodes **MouseDown**, **MouseMove** et **MouseUp** déclarées dans **TControl** (nous les déclarerons dans la section **protected**) :

```
protected
  procedure MouseDown(Button : TMouseButton; Shift : TShiftState; X, Y : integer); override;
  procedure MouseMove(Shift : TShiftState; X, Y : integer); override;
  procedure MouseUp(Button : TMouseButton; Shift : TShiftState; X, Y : integer); override;
```

La première devra créer le clone et le positionner de façon centrée par rapport à la souris. La deuxième devra, si un clone existe, le déplacer à la nouvelle position de la souris. La troisième devra - de nouveau seulement si un clone existe - le détruire et envoyer un événement **OnDrop** en fonction de la position de la souris.

Puisque deux routines doivent positionner le clone de la même façon, nous créerons un méthode privée (**private**) qui s'occupera de cette tâche :

```
private
  procedure CalcClonePos(X, Y : integer);
```

Il reste un petit point à éclaircir : le clone ayant pour parent le parent du **TDropImage**, et les coordonnées reçues étant en rapport avec le **TDropImage**, nous devons trouver un moyen de calculer les coordonnées par rapport au parent. Cela peut être fait avec la méthode ClientToParent de **TControl**.

De même, nous utiliserons cette méthode pour calculer les coordonnées où l'on a déposé l'image si la propriété **DropControl** est à **nil**, et une combinaison des méthodes ClientToScreen et ScreenToClient dans le cas contraire.

Voici donc l'implémentation de ces quatre méthodes :

```
procedure TDropImage.CalcClonePos(X, Y : integer);
var Pt : TPoint;
begin
  Pt := ClientToParent(Point(X, Y));
  FClone.Left := Pt.X - Width  div 2;
  FClone.Top  := Pt.Y - Height div 2;
end;

procedure TDropImage.MouseDown(Button : TMouseButton; Shift : TShiftState; X, Y : integer);
begin
  inherited;
  if (not Enabled) or (Button <> mbLeft) then exit;
  if FClone <> nil then // par précaution, mais ne devrait jamais arriver
    FClone.Free;

  // Création du clone
  FClone := TImage.Create(Self);
  FClone.Parent := Parent;
  FClone.Width := Width;
  FClone.Height := Height;
  FClone.Stretch := Stretch;
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://sjrd.developpez.com/delphi/tutoriel/composants/heritage/

```
    FClone.Picture.Assign(Picture);

    // Calcul de la position initiale du clone
    CalcClonePos(X, Y);
  end;

  procedure TDropImage.MouseMove(Shift : TShiftState; X, Y : integer);
  begin
    inherited;
    if FClone <> nil then
      CalcClonePos(X, Y);
  end;

  procedure TDropImage.MouseUp(Button : TMouseButton; Shift : TShiftState; X, Y : integer);
  var Pt : TPoint;
  begin
    inherited;
    if FClone = nil then exit;

    // Destruction du clone
    FreeAndNil(FClone);

    // Calcul des coordonnées à envoyer à l'événement OnDrop
    if not Assigned(FOnDrop) then exit;
    Pt := Point(X, Y);
    if FDropControl = nil then Pt := ClientToParent(Pt) else
    begin
      Pt := FDropControl.ScreenToClient(ClientToScreen(Pt));

      // Dans ce cas on vérifie que l'on a bien déposé sur le contrôle DropControl
      if not PtInRect(Rect(0, 0, DropControl.Width, DropControl.Height), Pt) then
        exit;
    end;
    FOnDrop(Self, Pt.X, Pt.Y);
  end;
```

Un petit bout de ce code peut paraître étrange :

```
  if FClone <> nil then // par précaution, mais ne devrait jamais arriver
    FClone.Free;
```

Normalement, puisqu'on libère le clone dans la méthode **MouseUp**, **FClone** devrait toujours valoir **nil** à ce moment. Cependant, il se peut, et cela est indépendant de notre volonté, que l'événement de relâchement de la souris se *perde*, par exemple si une autre application apparaît devant la nôtre. Donc, on libère le clone par précaution, bien qu'en théorie nous ne devrions pas le faire.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://sjrd.developpez.com/delphi/tutoriel/composants/heritage/

VII - Sécuriser la destruction du contrôle DropControl

Un gros problème de sécurité s'est glissé dans la création du **TDropImage** : que se passera-t-il en effet si le contrôle référencé par **DropControl** est détruit ?

En exécution, rien de bien exceptionnel : vous obtiendrez une erreur de type **EAccessViolation** au moment de lâcher l'image.

En conception par contre, c'est Delphi qui se crache : plus moyen de faire quoi que ce soit, il est trop tard pour enregistrer, vous devez fermer Delphi, ou pire, le *killer* !

Pourtant, si par exemple vous ajoutez un **TActionList** et un **TImageList**, et que vous associer **ImageList1** à **ActionList1.Images**, avant de supprimer **ImageList1**, il ne se passe rien de grave et mieux : la référence est automatiquement supprimée !

Pour faire cela, le **TActionList** se base sur le système de *notification* de destruction. Chaque composant possède une liste de notification de destruction, et au moment d'être détruit, il appelle la méthode Notification déclarée dans **TComponent** avec comme paramètre **Operation** la valeur **opRemove**.

Pour s'ajouter dans la liste de notification d'un composant, il faut appeler sa méthode FreeNotification avec **Self** en paramètre. Pour s'en retirer, il faut appeler la méthode RemoveFreeNotification.

Ajoutez donc une méthode privée **SetDropControl** qui servira de méthode d'accès en écriture à la propriété **DropControl** (n'oubliez pas de modifier la clause **write** en conséquence) :

```
procedure TDropImage.SetDropControl(New : TControl);
begin
  if New = FDropControl then exit;
  if Assigned(FDropControl) and (not (csDestroying in FDropControl.ComponentState)) then
    FDropControl.RemoveFreeNotification(Self);
  FDropControl := New;
  if Assigned(FDropControl) then
    FDropControl.FreeNotification(Self);
end;
```

*La vérification concernant la présence du drapeau **csDestroying** dans l'ensemble FDropControl.ComponentState sert à éviter de modifier la liste de notifications du contrôle pendant que celui-ci la lit, ce qu'il ne fait qu'une fois que **csDestroying** est ajouté à cet ensemble.*

Comme vous pouvez le constater, cette méthode se charge de se supprimer de la liste de notification de destruction de l'ancien **DropControl**, puis s'ajoute à celle du nouveau.

Pour s'assurer qu'on ne laissera pas de trace dans la liste de notification du dernier **DropControl** après la destruction, nous ajouterons cette ligne de code au destructeur de **TDropImage** :

```
DropControl := nil;
```

Finalement, nous devons encore surcharger la méthode **Notification** pour passer **FDropControl** à **nil** le cas échéant :

```
protected
  procedure Notification(AComponent : TComponent; Operation : TOperation); override;
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://sjrd.developpez.com/delphi/tutoriel/composants/heritage/

Implémentez cette méthode comme suit :

```
procedure TDropImage.Notification(AComponent : TComponent; Operation : TOperation);
begin
  inherited;
  if (AComponent = FDropControl) and (Operation = opRemove) then
    FDropControl := nil;
end;
```

La méthode **Notification** héritée de **TComponent** appelle cette même méthode récursivement pour tous les composants qu'il possède (propriété **Components**).

Voici donc ce qui se passera lorsque le composant recensé par la propriété **DropControl** sera détruit :

1. Ce contrôle ajoute l'indicateur **csDestroying** à sa propriété **ComponentState**
2. Il appelle la méthode **Notification** que nous avons surchargée
3. Celle-ci passe à **nil** la propriété **DropControl**
4. **SetDropControl**, voyant l'indicateur **csDestroying**, n'appelle pas **RemoveFreeNotification**, mais la référence est bien supprimée

Nous avons maintenant terminé la création du composant **TDropImage**.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

VIII - Code complet du composant TDropImage

DropImg.pas

```
unit DropImg;

interface

uses
  Windows, SysUtils, Classes, Controls, ExtCtrls;

type
  TDropEvent = procedure(Sender : TObject; X, Y : integer) of object;

  TDropImage = class(TImage)
  private
    FClone : TImage;

    FDropControl : TControl;
    FOnDrop : TDropEvent;

    procedure SetDropControl(New : TControl);

    procedure CalcClonePos(X, Y : integer);
  protected
    procedure Notification(AComponent : TComponent; Operation : TOperation); override;
    procedure MouseDown(Button : TMouseButton; Shift : TShiftState; X, Y : integer); override;
    procedure MouseMove(Shift : TShiftState; X, Y : integer); override;
    procedure MouseUp(Button : TMouseButton; Shift : TShiftState; X, Y : integer); override;
  public
    constructor Create(AOwner : TComponent); override;
    destructor Destroy; override;
  published
    property DropControl : TControl read FDropControl write SetDropControl;
    property OnDrop : TDropEvent read FOnDrop write FOnDrop;
  end;

implementation

constructor TDropImage.Create(AOwner : TComponent);
begin
  inherited;
  FClone := nil;
  FDropControl := nil;
  FOnDrop := nil;
end;

destructor TDropImage.Destroy;
begin
  DropControl := nil;
  if FClone <> nil then
    FClone.Free;
  inherited;
end;

procedure TDropImage.SetDropControl(New : TControl);
begin
  if New = FDropControl then exit;
  if Assigned(FDropControl) and (not (csDestroying in FDropControl.ComponentState)) then
    FDropControl.RemoveFreeNotification(Self);
  FDropControl := New;
  if Assigned(FDropControl) then
    FDropControl.FreeNotification(Self);
end;

procedure TDropImage.CalcClonePos(X, Y : integer);
var Pt : TPoint;
begin
  Pt := ClientToParent(Point(X, Y));
  FClone.Left := Pt.X - Width  div 2;
  FClone.Top  := Pt.Y - Height div 2;
end;

procedure TDropImage.Notification(AComponent : TComponent; Operation : TOperation);
begin
  if (AComponent = FDropControl) and (Operation = opRemove) then
    FDropControl := nil;
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://sjrd.developpez.com/delphi/tutoriel/composants/heritage/

DropImg.pas

```
end;

procedure TDropImage.MouseDown(Button : TMouseButton; Shift : TShiftState; X, Y : integer);
begin
  inherited;
  if (not Enabled) or (Button <> mbLeft) then exit;
  if FClone <> nil then // par précaution, mais ne devrait jamais arriver
    FClone.Free;

  // Création du clone
  FClone := TImage.Create(Self);
  FClone.Parent := Parent;
  FClone.Width := Width;
  FClone.Height := Height;
  FClone.Stretch := Stretch;
  FClone.Picture.Assign(Picture);

  // Calcul de la position initiale du clone
  CalcClonePos(X, Y);
end;

procedure TDropImage.MouseMove(Shift : TShiftState; X, Y : integer);
begin
  inherited;
  if FClone <> nil then
    CalcClonePos(X, Y);
end;

procedure TDropImage.MouseUp(Button : TMouseButton; Shift : TShiftState; X, Y : integer);
var Pt : TPoint;
begin
  inherited;
  if FClone = nil then exit;

  // Destruction du clone
  FreeAndNil(FClone);

  // Calcul des coordonnées à envoyer à l'événement OnDrop
  if not Assigned(FOnDrop) then exit;
  Pt := Point(X, Y);
  if FDropControl = nil then Pt := ClientToParent(Pt) else
  begin
    Pt := FDropControl.ScreenToClient(ClientToScreen(Pt));

    // Dans ce cas on vérifie que l'on a bien déposé sur le contrôle DropControl
    if not PtInRect(Rect(0, 0, DropControl.Width, DropControl.Height), Pt) then
      exit;
  end;
  FOnDrop(Self, Pt.X, Pt.Y);
end;

end.
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://sjrd.developpez.com/delphi/tutoriel/composants/heritage/

IX - Test du composant

Nous en avons terminé avec la création du **TDropImage**. Vous pouvez le tester de la même façon que décrite en partie I, section III.

Dans le gestionnaire **OnCreate** de la fiche du projet **TestComposTuto.exe**, ajoutez le code suivant pour créer dynamiquement un composant de type **TDropImage** :

```
procedure TFormMain.FormCreate(Sender: TObject);
begin
  with TDropImage.Create(Self) do
  begin
    Parent := Self;
    Left := 8;
    Top := 8;
    AutoSize := True;
    Picture.LoadFromFile('Developpez-com.ico');
    OnDrop := DropImageDrop;
  end;
end;
```

Vous pouvez récupérer l'icône Developpez-com.ico à cet endroit.

Déclarez comme suit la procédure **DropImageDrop** dans la section **private** :

```
private
  { Déclarations privées }
  procedure DropImageDrop(Sender : TObject; X, Y : integer);
```

Et implémentez-là comme ci-dessous :

```
procedure TFormMain.DropImageDrop(Sender: TObject; X, Y: Integer);
begin
  ShowMessage(Format('Vous avez déposé l''image aux coordonnées (%d ; %d)', [X, Y]));
end;
```

Exécutez le programme de test, vous pouvez saisir l'image et la déposer n'importe où sur la fiche. Un message apparaît alors vous indiquant les coordonnées où vous l'avez déposée.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

X - Intégration dans la palette des composants

Rouvrez le package **ComposTutoD.bpl**, et son unité **RegComposTuto.pas**. Modifiez la routine **Register** comme suit pour intégrer le composant **TDropImage** dans la palette des composants (n'oubliez pas d'ajouter **DropImg** dans les **uses**) :

```
procedure Register;
begin
  RegisterComponents('Tutoriel', [TSelectDirDialog, TDropImage]);
end;
```

Compilez, puis rouvrez le projet **TestComposTuto.exe**. Le nouveau composant **TDropImage** est intégré à la palette. Vous pouvez voir qu'il a la même icône que le composant **TImage**, dont il hérite. Si vous voulez modifier l'icône, vous pouvez le faire en ajoutant une ressource de nom **TDROPIMAGE** dans **RegComposTuto.dcr** comme indiqué en partie I, section IV-G.

Vous pouvez le placer sur la fiche principale de **TestComposTuto**. Pour faire un test simple, n'affectez rien à la propriété **DropControl** et complétez comme suit le gestionnaire d'événements **OnDrop** :

```
procedure TFormMain.DropImageDrop(Sender: TObject; X, Y: Integer);
begin
  ShowMessage(Format('Vous avez déposé l''image aux coordonnées (%d ; %d)', [X, Y]));
end;
```

Pour l'image, vous pouvez reprendre l'icône de Développez.com utilisée dans le test par création dynamique.

Lancez l'application. Vous pouvez prendre l'image et la déposer quelque part sur la fiche.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

XI - Conclusion

Nous avons vu comment créer un composant visuel héritant d'un composant existant, en application un des concepts clefs de la POO, la réutilisation du code.

Nous avons également appris à utiliser les méthodes protégées déclarées dans les classes parentes afin d'intercepter les événements en interne, sans affecter les propriétés événements.

Vous pouvez télécharger tous les sources présentés dans ce tutoriel, tels qu'ils sont à ce stade.

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

Dans la prochaine section, nous parlerons de la réalisation de composants visuels *à partir de rien*, en héritant des classes abstraites de la VCL.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

XII - Liens utiles

Voici quelques liens utiles en rapport avec ce dont avons parlé dans cette partie :

- Comment créer ses propres composants
- Améliorer un panel pour ajouter du texte défilant

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

XIII - Remerciements

Je voudrais adresser un très grand MERCI à Laurent Dardenne pour sa grande aide dans la réalisation de ce tutoriel, autant aux niveaux documentation, fond et forme.

Merci aussi à gege2061 et Bestiol et encore une fois à Laurent Dardenne pour leurs relectures sur la forme et l'orthographe.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2005 - Sébastien Doeraene. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.