Chapitre 8 : les composants sont des logiciels réutilisables

8.1 Construction de composant avec Delphi

- Déivation à partir d'un composant visuel
- Construction par association de composants visuels
- Construction d'un composant non-visuel

8.2 Les messages Windows avec Delphi

- La programmation dirigée par les messages
- Mécanisme de la répartition des messages
- Création et envoi de messages

8.3 Création d'un événement associé à un message

- Rappel sur la construction d'un événement
- Exemple de création d'événements dans un TEdit

8.4 ActiveX avec la technologie COM

- Les notions d'interfaces COM de microsoft
- ActiveX est un objet COM
- Création d'un ActiveX avec Delphi
- Déploiement et utilisation Web d'une fiche ActiveX

Chapitre 8.1 Construction de composant avec Delphi

Plan du chapitre:

Introduction

1. Dérivation à partir d'un composant visuel

1.1 Ajout de méthodes à un composant d'arbre
1.2 Ajout de propriétés au composant d'arbre

2. Construire par association de composants visuels

2.1 Le composant de visualisation d'arbre TWinArbre
2.2 Evénement OnChange de TWinArbre
2.3 Evénement OnMouseDown de TWinArbre
2.4 Le composant final TWinArbre

2.4.1 Le composant TWinArbre peut se redimensionner

2.5 Le composant de saisie d'expression arithmétique

3. Construire un composant non visuel

3.1 Le composant TListe
3.2 Utilisation du composant TListe

Introduction

Ce chapitre est consacré à la production de composants logiciels réutilisables. Nous allons construire en Delphi trois genres d'exemples de " kits de logiciels " réutilisables et donc créer trois composants que vous pourrez améliorer ou modifier. Une fois terminé, chaque composant sera placé dans la palette des outils composants de Delphi ou Kylix (version Linux de Delphi).

Nous proposons encore une fois une démarche méthodique afin de construire certains de ces " kits ". Pour des composants visuels, nous nous limitons à la construction de composants par dérivation de composants existants. Nous montrons comment ajouter des propriétés ou des méthodes à un composant existant. Nous montrons aussi comment associer plusieurs composants visuels de Delphi pour construire un nouveau composant visuel. Nous construisons à la fin un composant non visuel.

1. Dérivation à partir d'un composant visuel

Nous allons construire un nouveau composant qui héritera d'une classe déjà existante de la VCL (Visual Composant Library), nous prendrons un composant visuel d'arbre. Notre action consistera essentiellement à étendre les fonctionnalités d'un composant déjà existant.

1.1 Ajout de méthodes à un composant d'arbre

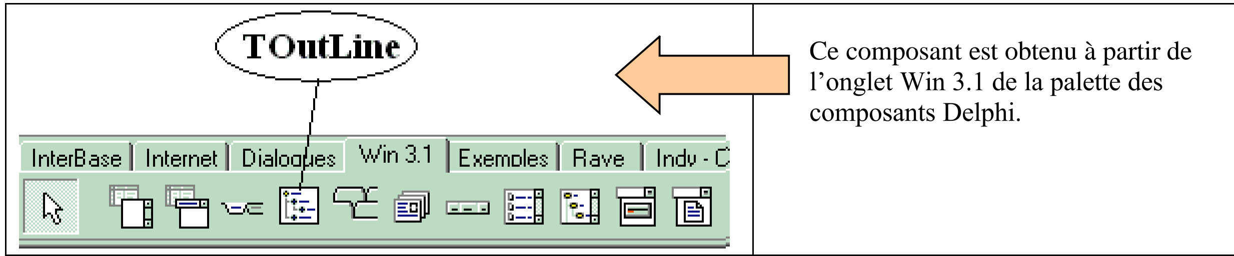
Démarche proposée en 3 étapes :

- Premier projet : **construire un programme** Delphi qui implante exactement les fonctionnalités de la nouvelle procédure (nouvelle action) et le tester.

- Deuxième projet : **construire une nouvelle classe** héritant du composant visuel existant, ajouter la nouvelle procédure qui vient d'être testée comme une méthode de la classe. Construire un programme de test de cette classe.

- Troisième projet : **transformer la classe en un composant**, l'installer dans la palette des composants, puis construire un programme de test du composant. Il suffira pour le programme de test de reprendre l'essentiel du programme de test de la classe.

Exemple : un composant d'arbre dérivé du TOutline



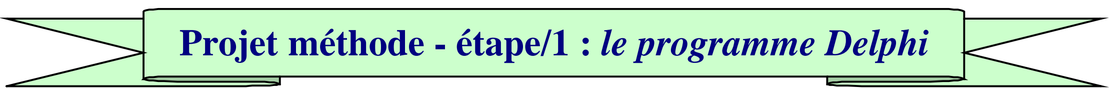
Ce composant est obtenu à partir de l'onglet Win 3.1 de la palette des composants Delphi.

Nous avons choisi ce composant pour deux raisons essentielles :

> 1° Sur le plan pédagogique c'est un bon outil simple de visualisation des structures d'arbres. Le lecteur pourra donc en suivant la démarche proposée ajouter ses propres méthodes de parcours d'arbre de tri etc...
>
> 2° Ce composant est présent dans toutes les versions de Delphi, ce qui le désigne comme candidat idéal à la dérivation pour nous (il est amélioré depuis par un composant TtreeView plus performant et moins simple à mettre en œuvre pour un débutant)

Fonctionnalité d'affichage de tout un niveau avec une méthode

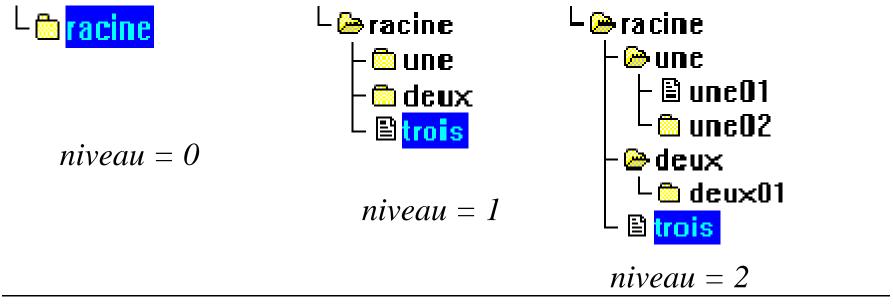
Nous désirons que notre composant affiche toutes les branches d'un arbre quelconque jusqu'à un niveau de profondeur donné. Appliquons la démarche précédente.

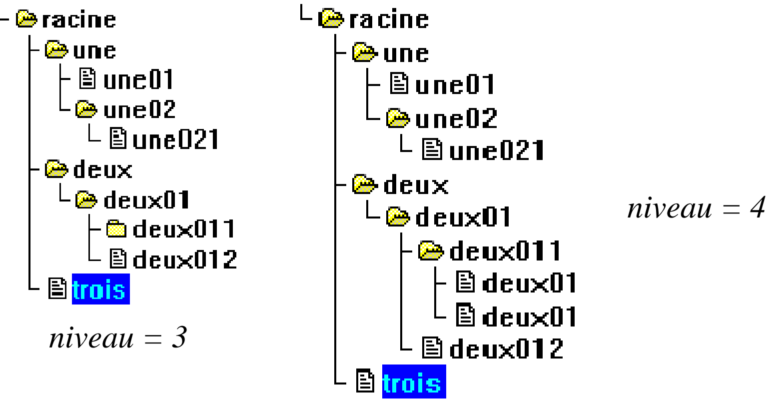


Nous écrivons 3 méthodes Delphi permettant d'effectuer cette action :

procedure affiche_racine (tree:Toutline)	*{remonte à la racine d'où que l'on soit}*
procedure lire_un_niveau (rac:Toutline;indice:integer; niveau:integer);	*{descente recursive en préordre sur un outline}*
procedure affiche_un_niveau (le_niveau:integer);	*{ visualise tout le niveau choisi par l'utilisateur, utilise les 2 méthodes précédentes, elle sera donc public }*

C'est la procédure affiche_un_niveau qui est appelée afin d'afficher l'arbre jusqu'au niveau voulu :





Projet méthode - étape/2 : *la classe Delphi*

On crée une classe Ttree2 dérivée du Toutline

```
TTree2 = class(TOutline) {la nouvelle classe TTree dérivée de Toutline}
private
 { Déclarations private }
 procedure lire_un_niveau (rac:Toutline; indice:integer; niveau:integer);
 procedure affiche_racine (tree:Toutline);
public
{ Déclarations public }
 procedure affiche_un_niveau (le_niveau:integer);
end;
```

On déclare un objet de classe Ttree2.

```
var
 new_compos : TTree2; {objet Ttree2 déclaré}
```

On instancie l'objet de classe Ttree2.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 new_compos := TTree2.create(self); {objet Ttree2 créé}
 new_compos.parent:=self; {objet Ttree2 affichable}
 new_compos.setbounds(8,8,233,233)
end;
```

On appelle la méthode public de l'objet.

```
   new_compos.affiche_un_niveau (3); {demande d' affichage niveau = 3}
```

Projet méthode - étape/3 : *le composant Delphi*

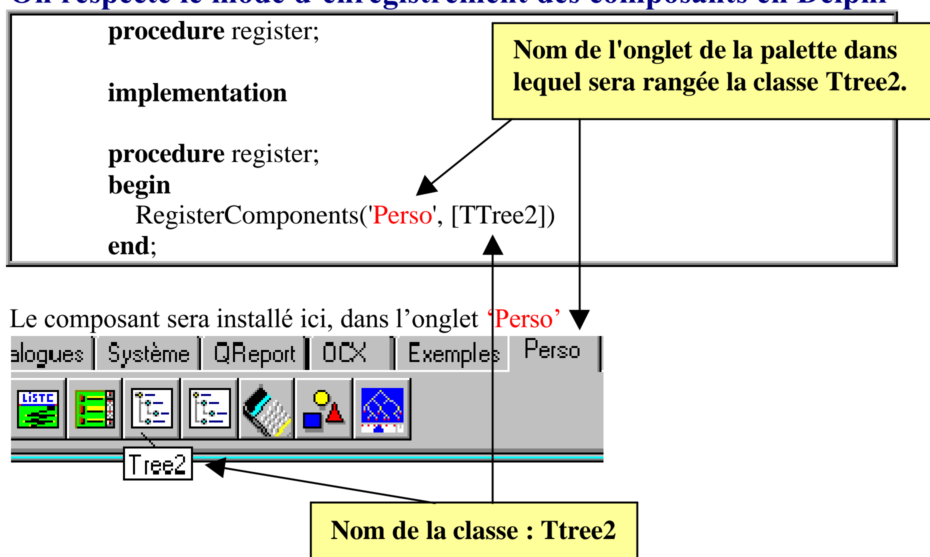
On ajoute un constructeur à la classe précédente Ttree2 :

```
TTree2 = class(TOutline) {la nouvelle classe TTree dérivée de Toutline}
private
  procedure lire_un_niveau (rac:Toutline; indice:integer; niveau:integer);
  procedure affiche_racine (tree:Toutline);
public
 constructor Create (Aowner:Tcomponent); override;
 procedure affiche_un_niveau(le_niveau:integer);
end;
```

Dans le constructeur nous reprenons le code du gestionnaire Oncreate de la fiche.

```
constructor TTree2.Create(Aowner:Tcomponent);
begin
   inherited create(Aowner);
   self.setbounds(8,8,233,233); {position : left,top,width,height}
end;
```

On respecte le mode d'enregistrement des composants en Delphi



Code complet du composant Ttree2

```
unit Utree2; { Un composant d'affichage d'arbre }
interface

 uses
  SysUtils, Messages, Classes, Graphics, Controls,  Forms, StdCtrls, ExtCtrls, Outline;

 type
  TTree2 = class(TOutline)
 private
  procedure affiche_racine(tree:TTree2);
  procedure TTree2.lire_un_niveau(rac:TTree2;  indice:integer;   le_niveau : integer);
 public
  constructor Create(Aowner:Tcomponent);override;
  procedure affiche_un_niveau (le_niveau : integer);
end;

  procedure register;

implementation

 procedure register;
 begin
  RegisterComponents('Perso',[TTree2])
 end;
 {---------------- Méthodes privées de la classe TTree2    ----------------}

 procedure TTree2.affiche_racine(tree:TTree2);
 {remonte à la racine d'où que l'on soit }
 var num_lev:integer;
```

```
begin
 if tree.Itemcount<>0 then
 begin
  num_lev:=tree.items[tree.selecteditem].topItem;
  tree.SelectedItem:=tree.items[num_lev].parent.index;
  tree.items[1].expanded:=false;
 end
end;

procedure TTree2.lire_un_niveau(rac:TTree2;indice:integer;le_niveau:integer);
{descente recursive en préordre sur un outline }
var
 node:ToutlineNode; {pour simplifier les manipulations}
 indice_node_fils,indice_node_pere:integer;
 begin
  if (indice<>-1)and(rac.Itemcount<>0) then
  begin
   node:=rac.items[indice];
   indice_node_pere:=rac.items[indice].parent.index;
   indice_node_fils:=indice;
   if node.HasItems then  {il y a des descendants}
   begin
    if node.level<=le_niveau then {uniquement si le niveau est correct}
    begin
     node.expand;             {visualiser les descendants à level+1}
     indice_node_pere:= rac.SelectedItem; {le noeud est le père}
     rac.SelectedItem:=rac.Items[rac.SelectedItem].GetFirstChild; {le 1er à gauche}
     indice_node_fils:=rac.SelectedItem;  {indice du noeud fils gauche}
     if indice_node_fils<>-1 then
     begin
      lire_un_niveau(rac,indice_node_fils,le_niveau);
      indice:=rac.Items[indice_node_pere].GetNextChild(indice_node_fils);{indice du frère suivant}
     end;
     while indice<>-1 do {examen de tous les frères de indice_node_fils}
     begin
      rac.SelectedItem:=indice;          {le frère suivant}
      indice_node_fils:=rac.SelectedItem; {le frère suivant est le nouveau fils}
      lire_un_niveau(rac,indice_node_fils,le_niveau);
      indice:=rac.Items[indice_node_pere].GetNextChild(indice_node_fils); {indice du frère suivant}
     end
    end
   end
  end;
 end;

{--------------------  Méthode public de la classe TTree2  ---------------------}
procedure TTree2.affiche_un_niveau(le_niveau:integer);
{pour visualiser tout le niveau choisi par l'utilisateur. }
var
 indice_noeud:integer;
 begin
  affiche_racine(self);
  if le_niveau<>0 then
  begin
   indice_noeud:= self.selecteditem;
   lire_un_niveau(self,indice_noeud,le_niveau);
   if self.Itemcount<>0 then
   begin
    indice_noeud:=self.Items[1].GetNextChild(indice_noeud);
    while indice_noeud<>-1 do
```

```
    begin
     self.selecteditem:=indice_noeud;
     lire_un_niveau(self,indice_noeud,le_niveau);
     indice_noeud:=self.Items[1].GetNextChild(indice_noeud);
    end
   end
  end
 end;

{////////////////////////  CONSTRUCTEUR  //////////////////////////}
constructor TTree2.Create(Aowner:Tcomponent);
begin
 inherited create(Aowner);
 self.setbounds(8,8,233,233); {position : left,top,width,height}
end;

end.
```

1.2 Ajout de propriétés au composant d'arbre

Démarche conseillée : ***semblable à la précédente***

- Premier projet : **construire un programme** Delphi qui implante exactement les fonctionnalités de la nouvelle procédure (nouvelle fonctionnalité) et le tester.
- Deuxième projet : **construire une nouvelle classe** héritant du composant visuel existant, et ajouter la nouvelle procédure qui vient d'être testée en la reliant à une propriété publique par exemple. Construire un programme de test de cette classe avec sa nouvelle propriété.
- Troisième projet : **transformer la classe en un composant**, l'installer dans la palette des composants, puis construire un programme de test du composant. Le composant a pratiquement été entièrement construit dans l'étape précédente.

Fonctionnalité d'affichage de tout un niveau avec une propriété

Nous désirons que notre composant affiche toutes les branches d'un arbre quelconque jusqu'à un niveau de profondeur donné à partir d'une propriété Delphi et non plus d'une méthode. Nous allons ainsi voir la puissance et la facilité fournies par le RAD.

Les propriétés ont en Delphi une particularité intéressante : celle de pouvoir être **lues** ou **écrites** à travers des méthodes internes que l'on peut programmer soi-même. Ceci nous donne une latitude importante lorsque nous voulons étendre les fonctionnalités d'une classe.

L'étape/1 est strictement la même que dans le traitement précédent sur l'ajout d'une nouvelle méthode :

procedure affiche_racine (tree:Toutline)	*{remonte à la racine d'où que l'on soit}*
procedure lire_un_niveau (rac:Toutline;indice:integer; niveau:integer);	*{descente recursive en préordre sur un outline}*
procedure affiche_un_niveau (le_niveau:integer);	*{ visualise tout le niveau choisi par l'utilisateur}*

Projet propriété - étape/2 : *la classe Delphi*

On crée une classe Ttree1 dérivée du Toutline.

```
TTree1 = class(TOutline) {la nouvelle classe Ttree1 dérivée de Toutline}
private
   Fniveau : integer;
   function Getmaxniveau : integer;
   procedure affiche_un_niveau(le_niveau:integer);
   procedure lire_un_niveau (rac:Toutline; indice:integer; niveau:integer);
   procedure affiche_racine (tree:Toutline);
public
   property profondeur : integer read Getmaxniveau;
   property show_niveau  :integer read Fniveau  write affiche_un_niveau;
end;
```

La classe possède deux propriétés :

- **property** profondeur
- **property** show_niveau

Etudions l'implantation de chacune d'elle :

property profondeur :

> Afin de montrer au lecteur les possibilités de lecture et d'écriture des propriétés, nous avons rajouté à la classe une nouvelle fonctionnalité : pouvoir consulter pour un arbre donné, la valeur de sa profondeur. Cette action est implantée à travers la propriété profondeur qui est en lecture seulement (puisqu'elle est uniquement consultable).

property profondeur : integer **read** Getmaxniveau;

La propriété profondeur lorsqu'elle sera lue lors de l'exécution fera appel à la méthode Getmaxniveau. Cette méthode doit être une fonction et doit renvoyer un résultat du même type que la propriété (ici un integer). La méthode interne Getmaxniveau renvoyant la profondeur de l'arbre, est construite par nos soins.

Voici un exemple de code possible :

```
function TTree1.Getmaxniveau:integer;
var i,max:integer;
begin
  if self.Itemcount < 0 then begin
   max:=1;
   for i:=1 to self.Itemcount do
     if max<self.Items[i].level then
        max:=self.Items[i].level;
    Getmaxniveau:=max-1;
  end
  else Getmaxniveau:=0 { profondeur racine=0}
end;
```

Par exemple, lorsqu'une instruction du genre 'x := profondeur' est exécutée, Delphi appellera la méthode Getmaxniveau qui fournira un résultat. Ce résultat sera lui-même automatiquement placé dans la variable x. Comme nous l'avons déjà vu, les propriétés ont la particularité d'être des champs que l'on peut lire à travers une fonction.

Cette particularité est intéressante puisque nous voyons dans l'exemple que c'est au moment où l'on appelle Getmaxniveau que le calcul de la profondeur est effectué. Il s'agit d'une action dynamique qui nous assure quelle que soit la modification apportée à l'arbre, nous aurons toujours sa profondeur réelle.

Les propriétés en Delphi sont donc comme des médias permettant d'accéder à des informations à travers elles.

property show_niveau

A titre d'exemple cette propriété est en lecture et écriture.
En écriture, elle est chargée de provoquer l'affichage de l'arbre sur tout un niveau fixé.
En lecture elle fournit la valeur du niveau de l'arbre actuellement affiché.

property show_niveau : integer **read** Fniveau **write** Affiche _un_niveau;

Elle possède la particularité d'être écrite à travers une méthode qui doit être en Delphi une procédure avec un seul paramètre transmis par adresse ou par valeur mais du même type que la propriété.

- Lorsque la propriété **show_niveau** est modifiée (donc en écriture) comme dans l'instruction "*show_niveau := x*", il est fait appel à la méthode interne *Affiche_un_niveau* à laquelle Delphi passe le paramètre x. Tout se passe comme si on avait écrit "*Affiche_un_niveau(x) ;"* . Nous avons donc un moyen de provoquer dynamiquement une action lorsque l'on modifie une propriété.

- Lorsque la propriété **show_niveau** est lue comme dans l'instruction "*x := show_niveau;*" , c'est en fait le contenu du champ privé Fniveau qui est lu et renvoyé dans la variable x. Tout se passe comme si on avait écrit " *x := Fniveau ;"*

Le champ privé Fniveau contient la valeur effective du numéro de niveau qui actuellement affiché par le Ttree1, cette valeur est mise à jour lorsqu'un changement d'affichage a lieu, en l'occurrence lors de l'appel de la méthode Affiche _un_niveau(x) qui devra assurer la transmission de la valeur x dans Fniveau.

On instancie et l'on crée l'objet de classe Ttree1

```
var
  new_compos : TTree1; {objet Ttree1 déclaré}
......
//On programme le code du gestionnaire de création de la fiche.
procedure TForm1.FormCreate(Sender: TObject);
begin
   new_compos:=TTree1.create(self); {objet TTree1 créé}
   new_compos.parent:=self; {objet TTree1 affichable}
   new_compos.setbounds(8,8,233,233); {position : left,top,width,height}
   new_compos.lines.loadfromfile('lines.txt'); {arbre chargé}
end;
```

Projet propriété - étape/3 : *le composant Delphi*

On ajoute un constructeur à la classe précédente Ttree1 :

```
TTree1 = class(TOutline) {la classe Ttree1 dérivée de Toutline}
private
   Fniveau:integer;
   function Getmaxniveau : integer;
   procedure affiche_un_niveau(le_niveau:integer);
   procedure lire_un_niveau (rac:Toutline; indice:integer; niveau:integer);
     procedure affiche_racine (tree:Toutline);
public
  constructor Create(Aowner:Tcomponent);override;
  property profondeur : integer read Getmaxniveau;
  property show_niveau : integer read Fniveau  write affiche_un_niveau;
end;
```

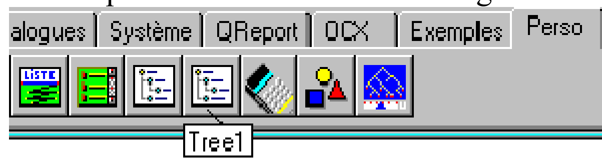
Identiquement à l'exemple précédent, nous remplaçons le code du gestionnaire de création de la fiche par le code du constructeur d'objets de la classe(Create).

```
constructor TTree1.Create(Aowner:Tcomponent);
begin
  inherited create(Aowner);
  self.setbounds(8,8,233,233); {position : left,top,width,height}
end;
```

On respecte le mode d'enregistrement des composants en Delphi

```
procedure register;

implementation

procedure register;
begin
 RegisterComponents('Perso', [TTree1])
end;
```

Le composant sera installé dans l'onglet 'Perso'à côté de Ttree2 :



Code complet du composant Ttree1

```
unit Utree1; { Un composant d'affichage d'arbre }
interface

 uses
  SysUtils, Messages, Classes, Graphics, Controls,
  Forms, StdCtrls, ExtCtrls, Outline;
```

```
type
 TTree1 = class(TOutline)  {la nouvelle classe Ttree1 de Toutline}
private
 Fniveau : integer;
 function Getmaxniveau:integer;
 procedure affiche_un_niveau(le_niveau:integer);
 function Getmaxniveau : integer;
 procedure affiche_un_niveau(le_niveau:integer);
 procedure lire_un_niveau (rac:Toutline; indice:integer; niveau:integer);
 procedure affiche_racine (tree:Toutline);
public
 constructor Create(Aowner:Tcomponent);override;
  property profondeur : integer read Getmaxniveau;
  property show_niveau : integer read Fniveau write affiche_un_niveau;
end;

 procedure register;

implementation

 procedure register;
 begin
  RegisterComponents('Perso',[TTree1])
 end;

 {-------------------- méthodes private du TTree1 -----------------------}

 procedure TTree1.affiche_racine(tree:TTree1);
 {remonte à la racine d'où que l'on soit }
 var num_lev:integer;
 begin
  if tree.Itemcount<>0 then
  begin
   num_lev:=tree.items[tree.selecteditem].topItem;
   tree.SelectedItem:=tree.items[num_lev].parent.index;
   tree.items[1].expanded:=false;
  end
 end;

 procedure TTree1.lire_un_niveau(rac:TTree1;indice:integer;le_niveau:integer);
 {descente recursive en préordre sur un outline}
 var
  node:ToutlineNode; {pour simplifier les manipulations}
  indice_node_fils,indice_node_pere:integer;
  begin
   if (indice<>-1)and(rac.Itemcount<>0) then
   begin
    node:=rac.items[indice];
    indice_node_pere:=rac.items[indice].parent.index;
    indice_node_fils:=indice;
    if node.HasItems then  {il y a des descendants}
    begin
     if node.level<=le_niveau then {uniquement si le niveau est correct}
     begin
      node.expand;             {visualiser les descendants à level+1}
      indice_node_pere:= rac.SelectedItem; {le noeud est le père}
      rac.SelectedItem:=rac.Items[rac.SelectedItem].GetFirstChild; {le 1er à gauche}
      indice_node_fils:=rac.SelectedItem;  {indice du noeud fils gauche}
      if indice_node_fils<>-1 then
```

```
     begin
      lire_un_niveau(rac,indice_node_fils,le_niveau);
      indice:=rac.Items[indice_node_pere].GetNextChild(indice_node_fils);{indice du frère suivant}
     end;
     while indice<>-1 do {examen de tous les frères de indice_node_fils}
     begin
      rac.SelectedItem:=indice;          {le frère suivant}
      indice_node_fils:=rac.SelectedItem; {le frère suivant est le nouveau fils}
      lire_un_niveau(rac,indice_node_fils,le_niveau);
      indice:=rac.Items[indice_node_pere].GetNextChild(indice_node_fils); {indice du frère
suivant}
     end
    end
   end
  end;
 end;

 procedure TTree1.affiche_un_niveau(le_niveau:integer);
 {pour visualiser tout le niveau choisi par }
 var
  indice_noeud:integer;
  begin
   affiche_racine(self);
   if le_niveau<>0 then
   begin
    indice_noeud:= self.selecteditem;
    lire_un_niveau(self,indice_noeud,le_niveau);
    if self.Itemcount<>0 then
    begin
     indice_noeud:=self.Items[1].GetNextChild(indice_noeud);
     while indice_noeud<>-1 do
     begin
      self.selecteditem:=indice_noeud;
      lire_un_niveau(self,indice_noeud,le_niveau);
      indice_noeud:=self.Items[1].GetNextChild(indice_noeud);
     end
    end
   end
  end;

 function TTree1.Getmaxniveau:integer;
 {donne la profondeur maximum de l'arbre }
 var i,max:integer;
 begin
  if self.Itemcount<>0 then
  begin
   max:=1;
   for i:=1 to self.Itemcount do
    if max<self.Items[i].level then
     max:=self.Items[i].level;
   Getmaxniveau:=max-1;
  end
  else
   getmaxniveau:=0   // profondeur racine=0
 end;

 {////////////////////////  CONSTRUCTEUR  //////////////////////////}
 constructor TTree1.Create(Aowner:Tcomponent);
 {remplace le create dans l'étape 1}
 begin
```

```
 inherited create(Aowner);
 self.setbounds(8,8,233,233); {position : left,top,width,height}
end;

end.
```

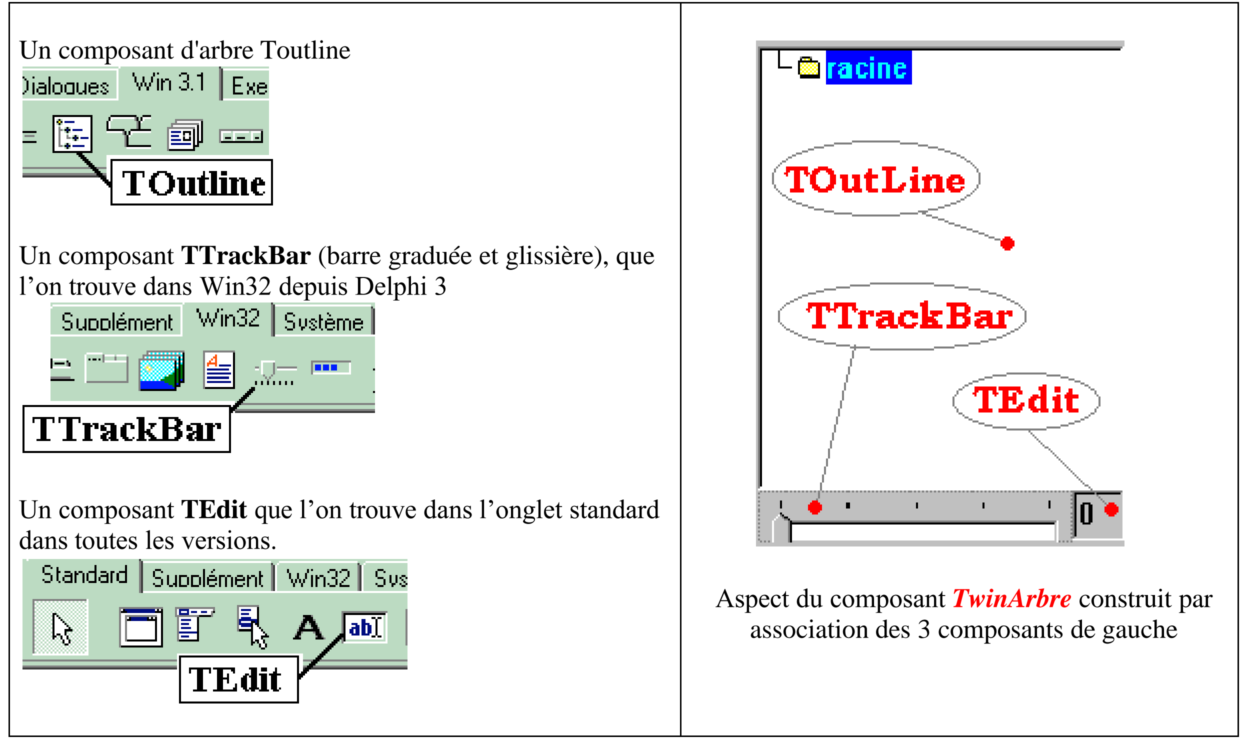
2. Construire par association de composants visuels

Nous nous proposons ici de fournir trois exemples de composants visuels construits par association (agrégation) d'autres composants visuels.

- Dans le premier exemple, nous montrons au lecteur comment associer trois composants visuels existants pour n'en former qu'un seul à partir du composant d'arbre fondé sur le TOutLine construit au paragraphe précédent.
- Dans le second exemple nous reprendrons la même démarche en construisant un composant plus élaboré de saisie des expressions arithmétiques en y incluant des résultats provenant des chapitres sur les analyseurs.
- le troisième exemple situé au paragraphe suivant, constitue la base de départ d'un composant d'éditeur-déboggeur d'un composant non visuel de liste de chaînes, à compléter à titre d'exercice.

2.1 Le composant de visualisation d'arbre

Nous montrons au lecteur comment associer trois composants visuels existants pour n'en former qu'un seul que nous notons ***TwinArbre***. Nous reprenons comme base le composant standard d'arbre, le **Toutline,** que nous associons à deux autres :



Un composant d'arbre Toutline

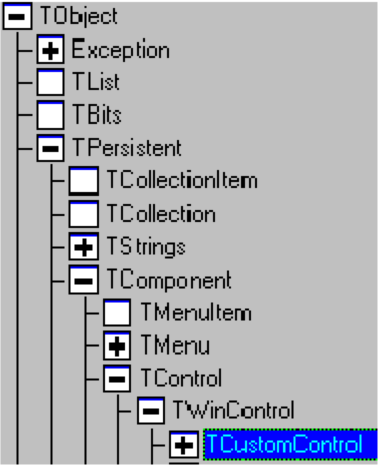
Un composant **TTrackBar** (barre graduée et glissière), que l'on trouve dans Win32 depuis Delphi 3

Un composant **TEdit** que l'on trouve dans l'onglet standard dans toutes les versions.

Aspect du composant ***TwinArbre*** construit par association des 3 composants de gauche

La démarche reste fondamentalement la même que celle que nous avons utilisée pour les deux exemples précédents. Ceci nous permet d'avoir une base simple, suffisante en initiation, d'élaboration de nouveaux composants.

Lorsque nous voulons construire un tel assemblage de composants il nous faut remonter dans la hiérarchie des classes. Delphi nous conseille de faire dériver notre futur composant ***TwinArbre*** systématiquement de la classe **TCustomControl** :



Nous vous livrons ci-après le composant ***TwinArbre*** en utilisant la version adjonction des propriétés. Nous élargissons les fonctionnalités par des nouvelles propriétés qui encapsulent des propriétés des 3 composants associés.

Propriétés du TwinArbre	Composant auquel elle est liée
property Potentiometre: TTrackBar **read** FPotentiometre;	La **property** Potentiometre est liée directement en lecture au composant TTrackBar.
property Tree: Toutline **read** FTree **write** FTree;	La **property** Tree est liée directement en lecture et en écriture au composant TOutline.
property Profondeur:integer **read** GetProfondeur;	La **property** profondeur est liée au composant TOutline.
property Enabled:boolean **read** Getenabled **write** Setenabled;	La **property** Enabled est reliée aux trois composants.
property Lignes: Tstrings **read** GetLignes **write** SetLignes;	La **property** Lignes est reliée au composant TOutline.
property Couleur: TColor **read** GetCouleur **write** SetCouleur;	La **property** Couleur est reliée au composant TOutline.
property Ascenceur: TScrollStyle **read** GetAscenceur **write** SetAscenceur ;	La **property** Ascenceur est reliée au composant TOutline.

Afin de donner une vue un peu plus élargie de la construction de tels composants, nous avons programmé deux événements dans notre composant ***TwinArbre*** : un événement associé au Toutline et un événement associé au TTrackBar. Nous "exportons" l'interception événementielle d'un des 3 composant associé comme étant un nouvel événement du composant **TCustomControl** formé par l'agrégation des 3 composants. La classe **TcustomControl** sert ainsi de classe "enveloppe".

Ci-dessous les codes sources de chaque propriété :

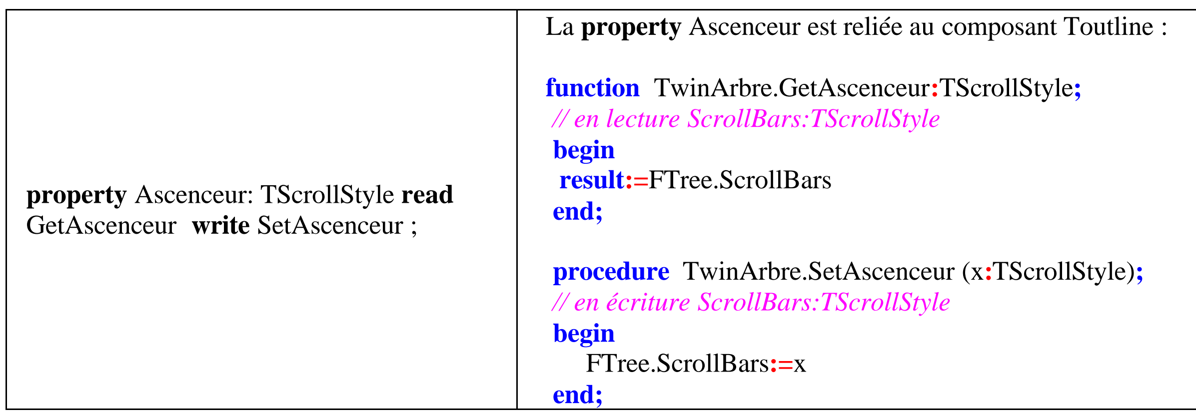
Propriétés du TwinArbre	Code Delphi de lecture/écriture

property Profondeur:integer
read GetProfondeur;
	La **property** profondeur est reliée au composant Toutline :
	function TwinArbre.GetProfondeur:integer;
	// met la profondeur de l'arbre dans la propriété profondeur
	begin
	result:=Getmaxniveau *// même métode que dans Ttree1*
	end;

property Enabled:boolean **read** Getenabled
write Setenabled;
	La **property** Enabled est reliée aux trois composants :
	function TwinArbre.GetEnabled:boolean;
	// en lecture enabled:Tboolean
	begin
	result:=FTree.enabled **and** FPotentiometre.enabled ;
	end;
	
	procedure TwinArbre.SetEnabled(x:boolean);
	// en écriture enabled:Tboolean
	begin
	FTree.Enabled:=x;
	FPotentiometre.enabled:=x ;
	if x=false **then**
	begin
	OldColor:=Ftree.color;
	Ftree.color:=clsilver;
	FEdit1.color:=clsilver
	end
	else
	begin
	Ftree.color:=OldColor;
	FEdit1.color:=OldColor
	end
	end;

property Lignes: Tstrings **read** GetLignes
write SetLignes;
	La **property** Lignes est reliée au composant Toutline:
	
	function TwinArbre.GetLignes:Tstrings;
	// en lecture lines:Tstrings
	begin
	result:=FTree.lines
	end;
	
	procedure TwinArbre.SetLignes(x:Tstrings);
	// en écriture lines:Tstrings
	begin
	FTree.lines:=x
	end;

property Couleur: TColor **read** GetCouleur
write SetCouleur;
	La **property** Couleur est reliée au composant Toutline :
	function TwinArbre.GetCouleur:TColor;
	// en lecture color:TColor
	begin
	result:=FTree.color
	end;
	
	procedure TwinArbre.SetCouleur(x:TColor);
	// en écriture color:TColor
	begin
	FTree.color:=x
	end;

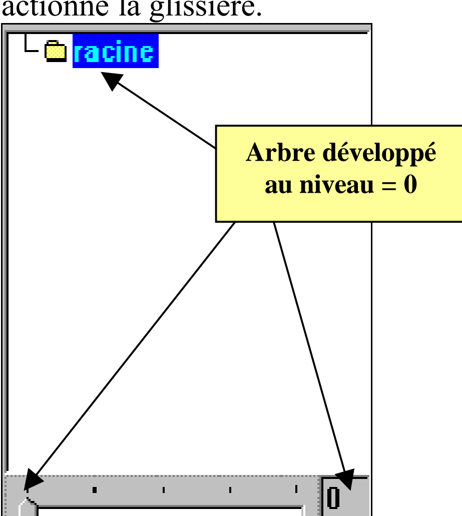


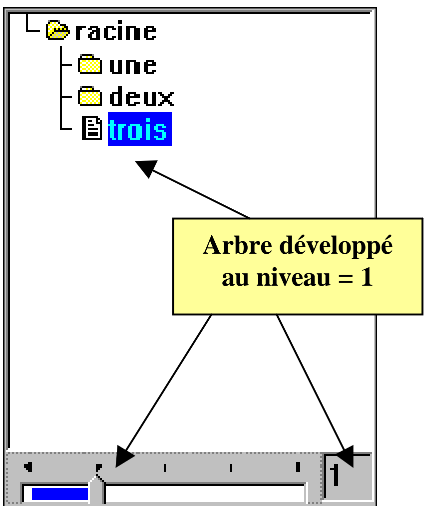
property Ascenceur: TScrollStyle **read** GetAscenceur **write** SetAscenceur ;	La **property** Ascenceur est reliée au composant Toutline :
	
	function TwinArbre.GetAscenceur:TScrollStyle;
	// en lecture ScrollBars:TScrollStyle
	begin
	result:=FTree.ScrollBars
	end;
	
	procedure TwinArbre.SetAscenceur (x:TScrollStyle);
	// en écriture ScrollBars:TScrollStyle
	begin
	FTree.ScrollBars:=x
	end;

Le lecteur pourra s'inspirer de ce développement pour écrire son code personnel sur d'autres événements.

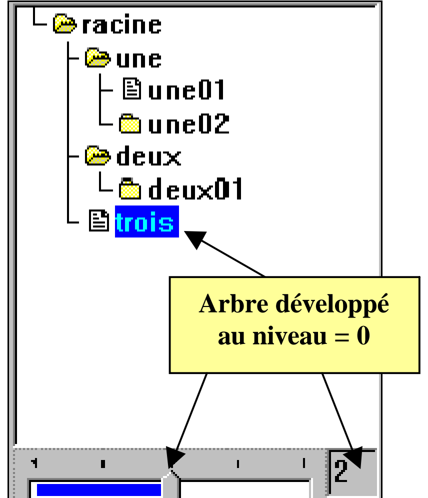
2.2 Evénement OnChange de TWinArbre

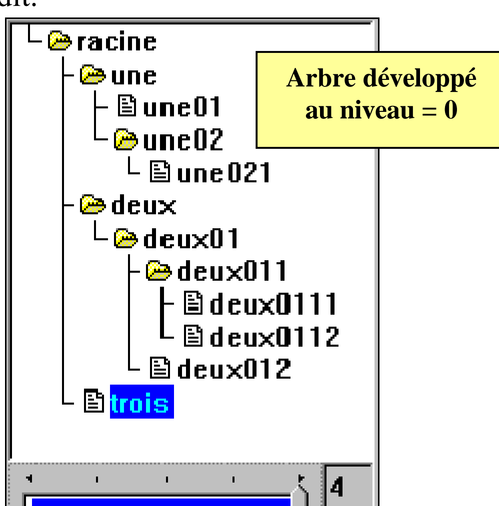
Nous avons programmé la réaction de notre composant à la manipulation de la glissière sur la barre graduée. Nous avons choisi l'événement **OnChange** du TTrackBar.
Cet événement permet d'afficher tout un niveau de l'arbre du Toutline lorsque l'utilisateur actionne la glissière.





La valeur du niveau est affichée dans le Tedit.





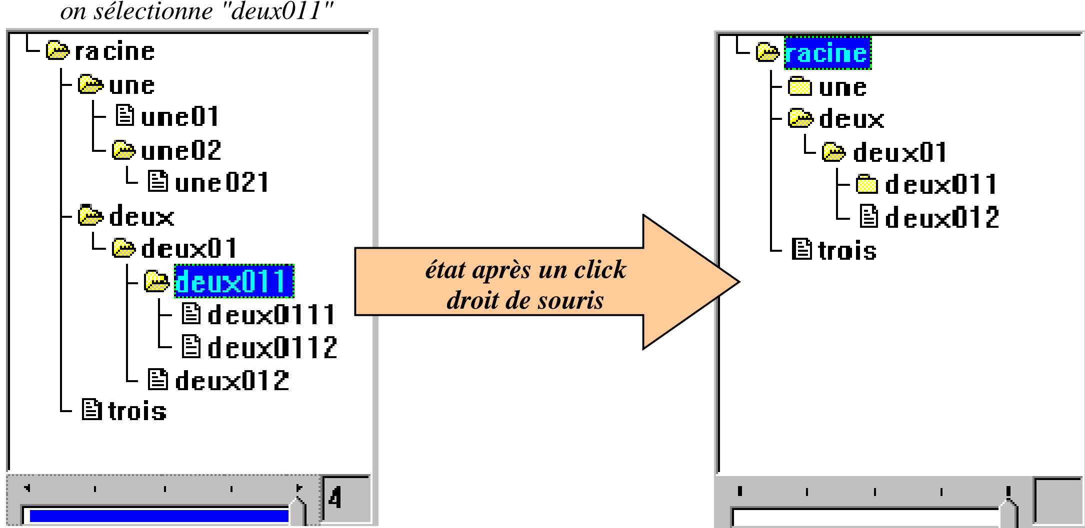
Le gestionnaire d'événement associé au niveau du **TCustomControl** doit avoir l'en-tête obligatoire d'un gestionnaire d'événement OnChange (un TNotifyEvent). Il a été nommé *PotentiometreChange*.Voici son en-tête :

```
procedure PotentiometreChange(Sender: TObject);
```

2.3 Evénement OnMouseDown de TWinArbre

Nous avons programmé de la même façon, une réaction spécifique de notre composant **TCustomControl** sur un click ***du bouton droit*** de la souris dans la zone du TOutline. Nous avons choisi pour ce faire l'événement **OnMouseDown** du Toutline.

Cet événement permet d'afficher ***seulement le chemin partant de la racine vers la feuille ou le noeud sélectionné*** ', le reste de l'arbre n'étant pas développé :



Le chemin **racine\deux\deux01\deux011** est affiché visuellement, toutes les autres branches inutiles visuellement sont refermées.

Le gestionnaire d'événement associé au niveau du **TCustomControl** a été nommé *ArbreMouseDown*. Il doit avoir l'en-tête obligatoire d'un gestionnaire d'événement **OnMouseDown** (un TmouseEvent). Voici son en-tête :

```
procedure ArbreMouseDown (Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
```

2.4. Le composant TWinArbre peut se redimensionner

En anticipant sur le prochain chapitre, nous avons rajouté, à l'intention du lecteur désireux de pouvoir modifier la taille de son composant lors de la conception avec son environnement Delphi, une méthode interne spécifique de redimensionnement. Les explications complètes du

fonctionnement, se trouvent au chapitre sur les messages Windows. Nous livrons tel quel le code de la méthode autorisant le redimensionnement du composant ***TwinArbre*** :

```
private
 procedure WMSize(var Message:TWMsize); message WM_SIZE;
 ......
procedure TwinArbre.WMSize(var Message:TWMsize);
{permet de modifier la taille du TCustomcontrol lors de la conception}
begin
   inherited;
   Ftree.setbounds(0,0,width,height-25);
   FPotentiometre.setbounds(0,FTree.Top+FTree.Height,width-25,25);
   FEdit1.setbounds(FPotentiometre.left+FPotentiometre.width,  Ftree.Top+Ftree.Height,25,25);
end;
```

Sans vouloir trop entrer dans de la technicité inutile et spécifique à ce RAD, indiquons que nous avons intercepté le message de modification de taille. Nous avons conçu tous les composants en positionnement relatif les uns par rapport aux autres en prenant comme référence de départ le Toutline Ftree. L'écriture " Ftree.setbounds(0,0,width,height-25) " indique que le Toutline est positionné en top=0, left=0 du composant, qu'il a toute la largeur du composant (Width) et que sa hauteur est celle du composant moins 25 pixels (height-25). Il faut donc dessiner sur le papier soigneusement le composant avant de l'implanter.

En fait cette attitude permet d'avoir une sorte d'homothétie sur les différents éléments visuels de ***TwinArbre***. Le lecteur pourra se servir de ce composant, il possède alors, une base de travail à enrichir soit par de nouvelles propriétés, soit par des réactions à de nouveaux événements.

Code complet du composant TWinArbre

```
unit UWinArbre;

interface
 uses
  SysUtils,WinTypes,WinProcs,Messages,Classes,Graphics,Controls,
  Forms, Dialogs, StdCtrls,ExtCtrls,Grids,Outline,ComCtrls;
 type

  TwinArbre = class(TCustomControl)
 private
 FPotentiometre: TTrackBar;
 FEdit1: TEdit;
 FTree:Toutline;
 OldColor:TColor;
  procedure WMSize(var Message:TWMsize);message WM_SIZE;
  function  Getmaxniveau:integer;
  function  GetProfondeur:integer;
  function  GetEnabled:boolean;
  procedure SetEnabled(x:boolean);
  function  GetLignes:Tstrings;
  procedure SetLignes(x:Tstrings);
  function  GetCouleur:TColor;
  procedure SetCouleur(x:TColor);
```

```
 function  GetAscenceur:TScrollStyle;
 procedure SetAscenceur(x:TScrollStyle);
 procedure affiche_un_niveau(le_niveau:integer);
 procedure affiche_racine;
 procedure lire_un_niveau(indice:integer;niveau:integer);
 procedure CalCulChemin(Noeud:ToutLineNode;Liste:TStringList);
 procedure ArbreMouseDown(Sender: TObject; Button: TMouseButton;
  shift: tshiftstate;
  x, y: integer);
 procedure PotentiometreChange(Sender: TObject);
public
 Liste: TStringList;
 constructor Create(Aowner:Tcomponent);override;
 property Potentiometre: TTrackBar read FPotentiometre;
 property Tree: Toutline read FTree write FTree;
 property Profondeur:integer read GetProfondeur;
published
 property Enabled:boolean read Getenabled write Setenabled;
 property Lignes: Tstrings read GetLignes write SetLignes;
 property Couleur: TColor read GetCouleur write SetCouleur;
 property Ascenceur: TScrollStyle read GetAscenceur write SetAscenceur;
end;

 procedure Register;

implementation

procedure Register;
begin
 RegisterComponents('Perso', [TwinArbre]);
end;
{ //////////////////// les constructeurs ///////////////////////}
procedure TwinArbre.WMSize(var Message:TWMsize);
{permet de modifier la taille du customcontrol lors de la conception}
begin
 inherited;
 Ftree.setbounds(0,0,width,height-25);
 FPotentiometre.setbounds(0,FTree.Top+FTree.Height,width-25,25);
 FEdit1.setbounds(FPotentiometre.left+FPotentiometre.width,Ftree.Top+Ftree.Height,25,25);
end;

constructor TwinArbre.Create(Aowner:Tcomponent);
begin
 inherited create(Aowner);
 OldColor:=clWindow;
 setbounds(10,10,200,200);
 Ftree:=Toutline.create(self);
 Ftree.parent:=self;
 Ftree.setbounds(0,0,width,height-25);
 FTree.OnMouseDown:=ArbreMouseDown;
 Ftree.color:=OldColor;
 FPotentiometre:=TTrackBar.create(self);
 FPotentiometre.parent:=self;
 FPotentiometre.setbounds(0,FTree.Top+FTree.Height,width-25,25);
 FEdit1:=TEdit.create(self);
 FEdit1.parent:=self;
 FEdit1.color:=OldColor;
 if Getmaxniveau>0 then
  FPotentiometre.Max:=Getmaxniveau-1
 else
```

```
 fpotentiometre.max:=0;
 FPotentiometre.Min:=0;
 FPotentiometre.Position:=0;
 FPotentiometre.LineSize:=1;
 FPotentiometre.PageSize:=1;
 FPotentiometre.TickMarks:=tmTopLeft;
 FPotentiometre.OnChange:=PotentiometreChange;
 FEdit1.setbounds(FPotentiometre.left+FPotentiometre.width,Ftree.Top+Ftree.Height,25,25);
 FEdit1.text:=inttostr(FPotentiometre.Min);
 FEdit1.ReadOnly:=true;
 Liste:=TStringList.create;
end;
{/////////////////////// implantation //////////////////////}

function TwinArbre.Getmaxniveau:integer;
{donne la profondeur maximum}
var i,max:integer;
begin
 if Ftree.Itemcount<>0 then
 begin
  max:=1;
  for i:=1 to Ftree.Itemcount do
   if max<Ftree.Items[i].level then
    max:=Ftree.Items[i].level;
  Getmaxniveau:=max;
 end
 else
  getmaxniveau:=0
end;

procedure TwinArbre.affiche_racine;
{remonte à la racine d'où que l'on soit}
var num_lev:integer;
begin
 if Ftree.Itemcount<>0 then
 begin
  num_lev:=Ftree.items[Ftree.selecteditem].topItem;
  Ftree.SelectedItem:=Ftree.items[num_lev].parent.index;
  Ftree.items[1].expanded:=false;
 end
end;

procedure TwinArbre.lire_un_niveau(indice:integer;niveau:integer);
{descente recursive en préordre sur un outline}
var node:ToutlineNode; {pour simplifier les manipulations}
 indice_node_fils,indice_node_pere:integer;
 begin
  if (indice<>-1)and(Ftree.ItemCount<>0) then
  begin
   node:=Ftree.items[indice];
   indice_node_pere:=Ftree.items[indice].parent.index;
   indice_node_fils:=indice;
   if node.HasItems then  {il y a des descendants}
   begin
    if node.level<=niveau then {uniquement si le niveau est correct}
    begin
     node.expand;            {visualiser les descendants à level+1}
     indice_node_pere:= Ftree.SelectedItem; {le noeud est le père}
     Ftree.SelectedItem:=Ftree.Items[Ftree.SelectedItem].GetFirstChild; {le 1er à gauche}
     indice_node_fils:=Ftree.SelectedItem;  {indice du noeud fils gauche}
```

```
      if indice_node_fils<>-1 then
      begin
       lire_un_niveau(indice_node_fils,niveau);
       indice:=Ftree.Items[indice_node_pere].GetNextChild(indice_node_fils);{indice du frère suivant}
      end;
      while indice<>-1 do {examen de tous les frères de indice_node_fils}
      begin
       Ftree.SelectedItem:=indice;          {le frère suivant}
       indice_node_fils:=Ftree.SelectedItem; {le frère suivant est le nouveau fils}
       lire_un_niveau(indice_node_fils,niveau);
       indice:=Ftree.Items[indice_node_pere].GetNextChild(indice_node_fils); {indice du frère suivant}
      end
     end
    end
   end
  end;

 procedure TwinArbre.affiche_un_niveau(le_niveau:integer);
 {pour visualiser tout le niveau choisi par l'utilisateur}
 var
  indice_noeud:integer;
  begin
   affiche_racine; {affiche la racine dans tous les cas}
   if le_niveau<>0 then
   begin
    indice_noeud:= Ftree.selecteditem;
    lire_un_niveau(indice_noeud,le_niveau);
    if Ftree.Itemcount<>0 then
    begin
     indice_noeud:=Ftree.Items[1].GetNextChild(indice_noeud);
     while indice_noeud<>-1 do
     begin
      Ftree.selecteditem:=indice_noeud;
      lire_un_niveau(indice_noeud,le_niveau);
      indice_noeud:=Ftree.Items[1].GetNextChild(indice_noeud);
     end
    end
   end
  end;

 procedure TwinArbre.CalCulChemin(Noeud:ToutLineNode;Liste:TStringList);
 {stockage dans une Liste du chemin des numéros
de noeuds depuis la racine jusqu'à "noeud"  }
 begin
  if Noeud.index<>1 then {on ne remonte pas au delà de la racine!}
  begin
   Liste.add(inttostr(Noeud.parent.index));
   CalCulChemin(Noeud.parent,Liste)
  end
 end;
 {---------------- Les gestionnaires d'événements -------------}

 procedure TwinArbre.PotentiometreChange(Sender: TObject);
 {le curseur sert à définir le n° du niveau de l'arbre à afficher}
 begin
  if Getmaxniveau>0 then
   FPotentiometre.Max:=Getmaxniveau-1
  else
   fpotentiometre.max:=0;
  FEdit1.text:=inttostr(FPotentiometre.Position);
```

```
 FPotentiometre.Selstart:=0;
 FPotentiometre.Selend:=FPotentiometre.Position;
 affiche_un_niveau(FPotentiometre.Position);
end;

procedure TwinArbre.ArbreMouseDown(Sender: TObject; Button: TMouseButton;
{lorsqu'on clicke avec le bouton droit de souris l'objet émet un son
et il affiche uniquement le chemin permettant d'arriver à l'élément
qui est actuellement sélectionné dans l'arbre. }
shift: tshiftstate;
x, y: integer);
var numItem,i:integer;
begin
 with Sender as TOutLine do
 begin
  numItem:=selecteditem;
  if numItem>0 then {-1 pour rien de sélectionné et 0 si vide}
   if Button=mbRight then
   begin
    MessageBeep(MB_ICONASTERISK); {function de l'API Windows: émet un son}
    Liste.clear;
    CalCulChemin(items[numItem],Liste);
    items[1].Collapse;          {fermeture de tout l'arbre}
    for i:=Liste.count-1 downto 0 do
     Items[strtoint(Liste.strings[i])].expand; {expansion des parents seulement}
    FEdit1.text:='';           {on montre que cette partie n'est pas active}
    FPotentiometre.Selstart:=0; {     ---   idem    ---    }
    FPotentiometre.Selend:=0;   {     ---   idem    ---    }
   end
 end
end;
{////////////////////// LES PROPRIETES ///////////////////////////}

function TwinArbre.GetProfondeur:integer;
// met la profondeur de l'arbre dans la propriété profondeur
begin
 result:=Getmaxniveau
end;

{----------  les propriétés héritées des composants utilisés ---------}

function  TwinArbre.GetLignes:Tstrings; // en lecture lines:Tstrings
begin
 result:=FTree.lines
end;

procedure  TwinArbre.SetLignes(x:Tstrings); // en écriture lines:Tstrings
begin
 FTree.lines:=x
end;

function  TwinArbre.GetCouleur:TColor; // en lecture color:TColor
begin
 result:=FTree.color
end;

procedure  TwinArbre.SetCouleur(x:TColor); // en écriture color:TColor
begin
 FTree.color:=x
end;
```

```
function  TwinArbre.GetAscenceur:TScrollStyle;  // en lecture ScrollBars:TScrollStyle
begin
 result:=FTree.ScrollBars
end;

procedure  TwinArbre.SetAscenceur(x:TScrollStyle);  // en écriture ScrollBars:TScrollStyle
begin
 FTree.ScrollBars:=x
end;

function  TwinArbre.GetEnabled:boolean;  // en lecture enabled:Tboolean
begin
 result:=FTree.enabled and FPotentiometre.enabled ;
end;

procedure  TwinArbre.SetEnabled(x:boolean);  // en écriture enabled:Tboolean
begin
 FTree.Enabled:=x;
 FPotentiometre.enabled:=x ;
 if x=false then
 begin
  OldColor:=Ftree.color;
  Ftree.color:=clsilver;
  FEdit1.color:=clsilver
 end
 else
 begin
  Ftree.color:=OldColor;
  FEdit1.color:=OldColor
 end
end;
end.
```

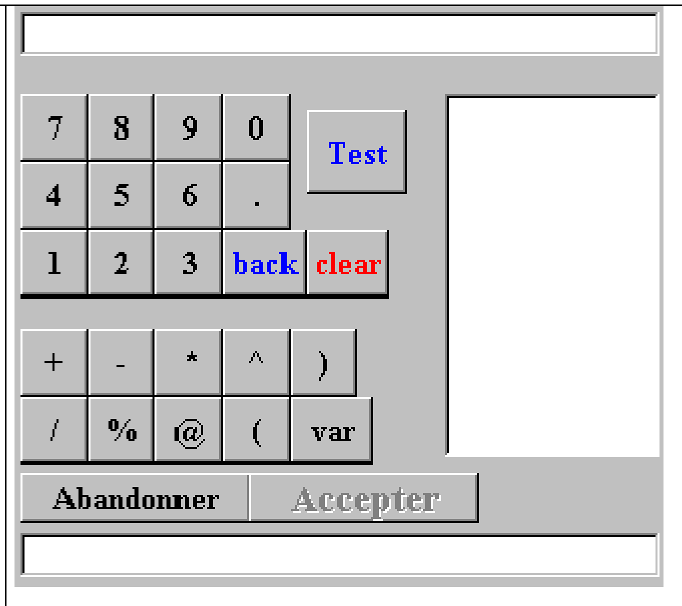
2.5 Un composant de saisie d'expression arithmétique avec Init et Follow

Voici le code d'un autre composant de saisie des expressions arithmétiques, élaboré à partir de plusieurs composants visuels associés.

Son développement met en œuvre la démarche du paragraphe précédent associée à une saisie par filtrage étudiée au chapitre de l'analyse des grammaires LL(1), avec un analyseur descendant récursif qui est inclu dans la unit du composant.

Le lecteur analysera les fonctions du logiciel et pourra y adjoindre de nouvelles fonctionnalités :

- Ajouter des boutons < , >, =, et les programmer.
- Ajouter des événements : OnTest, OnAccepter…



Code complet du composant TExpraritm

```
unit UComposExprarithm;
// composant de saisie d'expressions arithmétiques par filtrage
interface

 uses Controls,StdCtrls,Buttons,WinTypes,Classes;
 const
  et='\';
  ou='|';
  non='~';
  opdiv='%';
  opmod='@';
  Maxlongexpr=50;
  maxvar=50;
  LesChiffres='0123456789.';
  LesOper='+-*^)/%@(';
  LesCompar='<<>><='; // pour extension ultérieure
  ExprFausse='*****';
 type
  TypBouton=(T0,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,TTest,Tback,Tclear,Tplus,
  Tmoins,Tmult,Tpuiss,Tparferm,Tdiv,Tdivint,Tmod,Tparouvr,
  Tvar,Tleave,TFin);
  TableBouton=array[TypBouton]of TBitBtn;
  enscar=set of char;

  TExpraritm=class(TCustomControl)
 private
  Edit1: TEdit;
  Edit2: TEdit;
  ListBoxvar: TListBox;
  UnBouton:TableBouton;
  numcar: integer;
  carlu: char;
  LesFollow,LesInit:enscar;
  Chiffres,Lettres:enscar;
  edit_expr,expr_loc,expr_err:string; {expressions dans l'éditeur}
  erreur_expr:boolean;{erreur dans l'expression}
  procedure ListBoxClick(Sender: TObject);
  procedure BitBtnClick(Sender: TObject);
  procedure BitBtnTestClick(Sender: TObject);
  procedure BitBtnbackClick(Sender: TObject);
  procedure BitBtnclearClick(Sender: TObject);
  procedure BitBtnvarClick(Sender: TObject);
  procedure BitBtnleaveClick(Sender: TObject);
  procedure BitBtnfinClick(Sender: TObject);
  procedure expr;
  procedure init;
  procedure carsuiv;
  procedure err(n:integer);
  function  GetLignes:Tstrings;
  procedure SetLignes(x:Tstrings);
 public
  Expression:string;
  Reconnue:boolean;
  constructor create(Aowner:Tcomponent);override;
 published
  property Lignes:TStrings read GetLignes write SetLignes;
end;
```

```
 procedure Register;

implementation
uses Dialogs,SysUtils,Graphics;

 procedure Register;
 begin
  RegisterComponents('Perso', [TExpraritm]);
 end;
 {---------------- Utilitaires de positionnement-------------------}
 procedure PositionBoutonsChiffre(var Table:TableBouton);
 const h=33;
  w=33;
 var i:TypBouton;
 begin
  Table[T7].setbounds(8,48,H,W);
  Table[T8].setbounds(41,48,H,W);
  Table[T9].setbounds(74,48,H,W);
  Table[T0].setbounds(107,48,H,W);
  Table[T4].setbounds(8,81,H,W);
  Table[T5].setbounds(41,81,H,W);
  Table[T6].setbounds(74,81,H,W);
  Table[T10].setbounds(107,81,H,W);
  Table[T1].setbounds(8,114,H,W);
  Table[T2].setbounds(41,114,H,W);
  Table[T3].setbounds(74,114,H,W);
  for i:=T0 to T10 do
   Table[i].caption:=LesChiffres[ord(i)+1];
  {----------  Les autres boutons ---------}
  Table[Tback].setbounds(107,114,41,W);
  Table[Tback].caption:='back';
  Table[Tback].Font.color:=clBlue;
  Table[Tclear].setbounds(148,114,41,W);
  Table[Tclear].caption:='clear';
  Table[Tclear].Font.color:=clRed;
  Table[TTest].setbounds(148,56,49,41);
  Table[TTest].caption:='Test';
  Table[TTest].Font.color:=clBlue;
  Table[Tleave].setbounds(8,232,113,25);
  Table[Tleave].caption:='Abandonner';
  Table[TFin].setbounds(120,232,113,25);
  Table[TFin].caption:='Accepter';
  Table[TFin].Font.color:=clRed;
  Table[TFin].Font.Size:=14;
  Table[TFin].enabled:=false;
 end;

 procedure PositionBoutonsOper(var Table:TableBouton);
 const h=33;
  w=33;
 var i:TypBouton;
 begin
  Table[Tplus].setbounds(8,162,H,W);
  Table[Tmoins].setbounds(41,162,H,W);
  Table[Tmult].setbounds(74,162,H,W);
  Table[Tpuiss].setbounds(107,162,H,W);
  Table[Tparferm].setbounds(140,162,H,W);
  Table[Tdiv].setbounds(8,195,H,W);
  Table[Tdivint].setbounds(41,195,H,W);
```

```
 Table[Tmod].setbounds(74,195,H,W);
 Table[Tparouvr].setbounds(107,195,H,W);
 Table[Tvar].setbounds(140,195,41,W);
 for i:=Tplus to Tparouvr do
  Table[i].caption:=LesOper[ord(i)-ord(Tplus)+1];
 Table[Tvar].caption:='var';
end;
{///////////////////// la construction ////////////////////}
constructor TExpraritm.create(Aowner:Tcomponent);
var num:TypBouton;
begin
 inherited create(Aowner);
 setbounds(0,0,325,290);
 for num:= T0 to TFin do
 begin
  UnBouton[num]:=TBitBtn.create(self);
  UnBouton[num].parent:=self;
  UnBouton[num].Font.name:='Times New Roman';
  UnBouton[num].Font.Style:=[fsBold];
  UnBouton[num].Font.Size:=12;
  UnBouton[num].OnClick:=BitBtnClick;
 end;
 UnBouton[TTest].OnClick:=BitBtnTestClick;
 UnBouton[Tback].OnClick:=BitBtnbackClick;
 UnBouton[Tvar].OnClick:=BitBtnvarClick;
 UnBouton[Tleave].OnClick:=BitBtnleaveClick;
 UnBouton[TFin].OnClick:=BitBtnfinClick;
 UnBouton[Tclear].OnClick:=BitBtnclearClick;
 PositionBoutonsChiffre(UnBouton);
 PositionBoutonsOper(UnBouton);
 Edit1:=TEdit.create(self);
 Edit2:=TEdit.create(self);
 Edit1.parent:=self;
 Edit1.setbounds(8,8,313,27);
 Edit1.color:=clAqua;
 Edit1.ReadOnly:=true;
 Edit2.parent:=self;
 Edit2.setbounds(8,262,313,27);
 Edit2.color:=clYellow;
 ListBoxvar:=TListBox.create(self);
 ListBoxvar.parent:=self;
 ListBoxvar.setbounds(216,48,105,177);
 ListBoxvar.OnClick:=ListBoxClick;
 Expression:=ExprFausse;
 Reconnue:=false;
end;

{///////////  Les gestionnaires d'événements Onclick ///////////}
procedure TExpraritm.ListBoxClick(Sender: TObject);
begin
 if ListBoxvar.itemIndex<>-1 then
 begin
  expr_loc:=concat(expr_loc,ListBoxvar.items[ListBoxvar.itemIndex]);
  Edit1.text:=expr_loc
 end
end;

procedure TExpraritm.BitBtnClick(Sender: TObject);
var car:char;
begin
```

```
 with sender as TBitBtn do
 begin
  UnBouton[TFin].enabled:=false;
  car:=caption[1];
  if car in ['0'..'9']+['%','/','+','-','*','\','@','(',')','.','^'] then
   expr_loc:=concat(expr_loc,caption)
 end;
 edit1.text:=expr_loc;
end;

procedure TExpraritm.BitBtnTestClick(Sender: TObject);
begin
 Expression:=ExprFausse;
 Reconnue:=false;
 if length(expr_loc)<=Maxlongexpr then
 begin
  init;
  edit2.text:='';
  edit_expr:=expr_loc;
  edit_expr := concat(edit_expr,'#');
  carsuiv;
  erreur_expr:=false;
  expr;
  if erreur_expr then
   edit2.text:=expr_err
  else
  begin // expression correcte
   UnBouton[TFin].enabled:=true;
   Expression:=expr_loc
  end
 end
 else
  MessageDlg('Expression trop longue !', mtWarning,[mbOk], 0);
end;

procedure TExpraritm.BitBtnbackClick(Sender: TObject);
begin
 expr_loc:=copy(expr_loc,1,length(edit1.text)-1);
 edit1.text:=expr_loc;
 UnBouton[TFin].enabled:=false;
end;

procedure TExpraritm.BitBtnclearClick(Sender: TObject);
begin
 expr_loc:='';
 edit1.text:=expr_loc;
 UnBouton[TFin].enabled:=false;
end;

procedure TExpraritm.BitBtnvarClick(Sender: TObject);
begin
 if UnBouton[Tvar].tag=1 then
 begin
  ListBoxvar.visible:=true;
  UnBouton[Tvar].tag:=0;
 end
 else
 begin
  ListBoxvar.visible:=false;
  UnBouton[Tvar].tag:=1;
```

```
 end
end;

procedure TExpraritm.BitBtnleaveClick(Sender: TObject);
begin
 edit_expr:=ExprFausse;
 UnBouton[TFin].enabled:=false;
 Reconnue:=false;
 MessageBeep(MB_ICONASTERISK); {function de l'API Windows}
 Edit2.text:='Ok abandon reconnu!'
end;

procedure TExpraritm.BitBtnfinClick(Sender: TObject);
begin
 Reconnue:=true;
 MessageBeep(MB_ICONQUESTION); {function de l'API Windows}
 Edit2.text:='Ok expression acceptée !'
end;
{/////////////////////   Les Propriétés   //////////////////////}
function  TExpraritm.GetLignes:Tstrings;
begin
 result:=ListBoxvar.items
end;

procedure TExpraritm.SetLignes(x:Tstrings);
begin
 ListBoxvar.items:=x
end;

{//////////  Analyseur descendant récursif d'expressions ////////////}
procedure TExpraritm.init;
begin
 numcar := 0;
 LesFollow:=['+', '-', '*', '/','@','%',')','#','^'];
 LesInit:=['(','^','a'..'z','0'..'9'];
 Chiffres:=['0'..'9'];
 Lettres:=['a'..'z'];
end;

procedure TExpraritm.carsuiv;
Var CharS:string[1];
begin
 numcar := numcar + 1;
 CharS:=LowerCase(edit_expr[numcar]);
 carlu := CharS[1];
end;

procedure TExpraritm.err(n:integer);
var
 i: integer;
 begin
  expr_err:=concat('>>> Erreur ',inttostr(n),': ');
  Edit1.Hideselection:=false;
  Edit1.SelStart:=numcar-1; // sélection du car erroné
  Edit1.SelLength:=1;
  for i := 1 to numcar do
   expr_err:=concat(expr_err,edit_expr[i]);
  if carlu='#' then
   carlu:=edit_expr[numcar-1];
  expr_err:=concat(expr_err,' <--[ ',carlu,' ]');
```

```
 erreur_expr:=true;
end;

procedure TExpraritm.expr;
procedure fact;
begin
 if not erreur_expr then
 begin
  if carlu in LesInit then
  begin
   case
    carlu of
    '^':
    begin
     carsuiv;
     if carlu in LesInit then
     begin
      Fact;
      if erreur_expr then
       exit;
      carsuiv;
      if not(carlu in lesfollow) then
       err(7)
     end
     else
      err(8)
    end;
    '(':
    begin
     carsuiv;
     if carlu in LesInit then
     begin
      Expr;
      if erreur_expr then
       exit;
      if carlu<>')' then
       err(9)
      else
       carsuiv;
      if not(carlu in lesfollow) then
       err(10)
     end
     else
      err(11)
    end;
    'a'..'z':
    begin
     carsuiv;
     if carlu in Chiffres+Lettres then
      while carlu in Chiffres+Lettres do
       carsuiv;
     if not(carlu in lesfollow) then
      err(12)
    end;
    '0'..'9':
    begin
     carsuiv;
     while carlu in chiffres do
      carsuiv;
     if carlu='.' then
```

```
     begin
      carsuiv;
      while carlu in chiffres do
       carsuiv;
      if not(carlu in lesfollow) then
       err(13)
     end;
     if not(carlu in lesfollow) then
      err(14)
    end;
   end{case}
  end
 end
end;

procedure Terme;
begin
 if not erreur_expr then
 begin
  if carlu in LesInit then
  begin
   Fact;
   if erreur_expr then
    exit;
   while carlu in['*', '/','@','%','^'] do
   begin
    carsuiv;
    if carlu in lesinit then
     fact
    else
     err(1);
    if erreur_expr then
     exit;
   end;
   if not (carlu in lesfollow) then
    err(2)
  end
  else
   err(3)
 end
end;

begin{expr}
 if not erreur_expr then
 begin
  if carlu in LesInit then
  begin
   Terme;
   if erreur_expr then
    exit;
   while carlu in['+', '-'] do
   begin
    carsuiv;
    if carlu in lesinit then
     terme
    else
     err(4);
    if erreur_expr then
     exit;
   end;
```

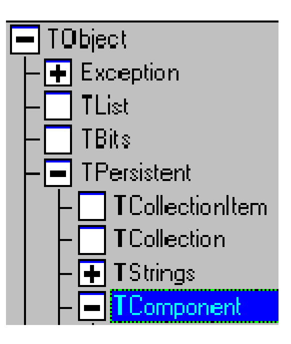
```
    if not (carlu in lesfollow) then
     err(5)
   end
   else
    err(6)
  end
 end;{expr}

end.
```

3. Construire un composant non visuel

En utilisant notre démarche de construction progressive et de tests successifs, nous pouvons sans effort particulier encapsuler dans un composant non visuel beaucoup d'unités réutilisables.

Nous allons encapsuler un TStringList dans un composant non visuel *construit à partir de la classe de composant abstrait TComponent* que nous nommons Tliste.

La seule différence avec les exemples précédents réside dans la classe de départ dont nous devons faire hériter notre futur composant. En Delphi nous devons dériver notre composant non visuel de la classe **Tcomponent**. La classe **TComponent** est le point de départ abstrait de tous les composants de Delphi et nous devons remonter à ce niveau lorsque nous voulons construire un composant qui ne sera pas un contrôle (un contrôle = un composant visuel).




```
TListe = class (TComponent)
        private
           FListGeneric:TstringList;
          ....
        public
           property ListGeneric:TstringList
                                read FListGeneric
                                write FListGeneric;
          ....
        end;
```

Comme nous voulions malgré tout bénéficier des propriétés et des méthodes de la classe TStringList, nous avons utilisé la démarche suivante :

- construire un champ privé du type dont on veut dériver (ici TstringList),
- construire une propriété du composant qui permettra de lire et d'écrire dans ce champ.

L'effort de construction que nous avons à apporter est minimal puisqu'il concerne uniquement l'exportation des méthodes

Code complet du composant TListe

Les propriétés et les méthodes sont fournies à titre d'exemple pédagogique. Nous encourageons le lecteur à réécrire et à développer lui-même à partir du composant TListe un composant personnalisé de liste de chaînes.

```
unit UListeCompos;

 interface

 uses
  StdCtrls,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

 type
  TListe = class(TComponent)
 private
  FListGeneric:TstringList;
  function  Getcount:integer;
  function  GetDuplic:boolean;
  procedure SetDuplic(x:boolean);
  function  Getsorted:boolean;
  procedure Setsorted(x:boolean);

 public
  constructor create(Aowner:Tcomponent);override;
  destructor  Liberer;
  {opérateurs de manipulation de la liste}
  procedure Effacer;
  function  Element(rang:integer):string;
  function  Position(element:string):integer;
  procedure Ajouter(ch:string);
  procedure Inserer(element:string;rang:integer);
  procedure Modifier(rang:integer;element:string);
  procedure Supprimer(rang:integer);
  {opérateurs de recopie d'une liste}
  procedure EstUneCopiede(T:TListe);
  procedure VersListBox(LBox:TListBox);
  {opérateurs d'entrée/sortie dans une liste}
  procedure Charger;
  procedure Sauver;
  {propriétés publiques, uniquement consultables}
  property Quantite:integer read Getcount ;
  property ListGeneric:TstringList read FListGeneric;

 published
  property Dupliquee:boolean read GetDuplic write SetDuplic;
  property Triee:boolean read Getsorted write Setsorted;
end;

  procedure Register;

implementation

 procedure Register;
 begin
  RegisterComponents('Perso', [TListe]);
```

```
end;

{//////////////////////////   PARTIE PUBLIC   ///////////////////////////}
constructor TListe.create(Aowner:Tcomponent);
begin
 inherited create(Aowner);
 FListGeneric:=TstringList.create;
 dupliquee:=false;
 triee:=false;
end;

destructor TListe.Liberer;
begin
 FListGeneric.free;
 inherited destroy;
end;
{///////-------------------- Méthodes publiques --------------------/////}
procedure TListe.Effacer;
{ré-initialise à vide}
begin
 FListGeneric.clear;
end;

function TListe.Element(rang:integer):string;
{fournit l'élément string de n°rang}
begin
 if rang in [1..FListGeneric.count] then
  Element:=FListGeneric.strings[rang-1]
 else
 begin
  Element:='';
  Application.MessageBox('>>> Méthode : ELEMENT',
  'Rang d''élément hors limites', mb_OK);
 end
end;

function TListe.Position(element:string):integer;
{ donne le rang de la première apparition de l'élément:
   position=0 si non l'élément n'est pasprésent dans la liste }
begin
 Position:=FListGeneric.IndexOf(element)+1
end;

procedure TListe.Ajouter(ch:string);
{ajout d'un élément en fin de liste}
begin
 FListGeneric.Add(ch);
end;

procedure TListe.Inserer(element:string;rang:integer);
{insérer l'élément donné au rang indiqué}
begin
 if not Triee then
  if rang in [1..FListGeneric.count] then
   FListGeneric.Insert(rang-1,element)
  else
   Application.MessageBox('>>> Méthode : INSERER',
 'Rang d''élément hors limites', mb_OK);
end;
```

```
procedure TListe.Modifier(rang:integer;element:string);
{changer le contenu au rang indiqué par l'élément donné}
begin
 if not Triee then
  if rang in [1..FListGeneric.count] then
   FListGeneric.strings[rang-1]:=element
  else
   Application.MessageBox('>>> Méthode : MODIFIER',
 'Rang d''élément hors limites', mb_OK);
end;

procedure TListe.Supprimer(rang:integer);
{supprimer l'élément situé à ce rang}
begin
 if rang in [1..FListGeneric.count] then
  FListGeneric.Delete(rang-1)
 else
  Application.MessageBox('>>> Méthode : SUPPRIMER',
 'Rang d''élément hors limites', mb_OK);
end;
{/////----------------- recopie d'une liste---------------------/////}

procedure TListe.EstUneCopiede(T:TListe);
{effectue une copie de liste}
begin
 Effacer;
 FListGeneric.Assign(T.ListGeneric);
end;

procedure TListe.VersListBox(LBox:TListBox);
{recopie en mode string le contenu de la liste dans une ListBox}
begin
 if assigned(LBox) then
  LBox.items:=FListGeneric
end;
{/////----------------------- ENTREE/SORTIE-----------------------/////}
procedure TListe.Charger;
{charger la liste à partir d'un fichier}
var Ouvrir:TOpenDialog;
begin
 Ouvrir:=TOpenDialog.create(self);
 Ouvrir.filter:='Texte|*.txt';
 Ouvrir.Title:='Chargement d''une liste';
 Ouvrir.Options:=[ofFileMustExist,ofHideReadOnly];
 if Ouvrir.execute then
  FListGeneric.LoadFromfile(Ouvrir.Filename);
 Ouvrir.free
end;

procedure TListe.Sauver;
{sauver la liste à partir dans un fichier}
var Save:TSaveDialog;
begin
 Save:=TSaveDialog.create(self);
 Save.filter:='Texte|*.txt';
 Save.Title:='Sauvegarder la liste';
 Save.Options:=[ofOverwritePrompt,ofHideReadOnly];
 if Save.execute then
  FListGeneric.SaveTofile(Save.Filename);
 Save.free
```

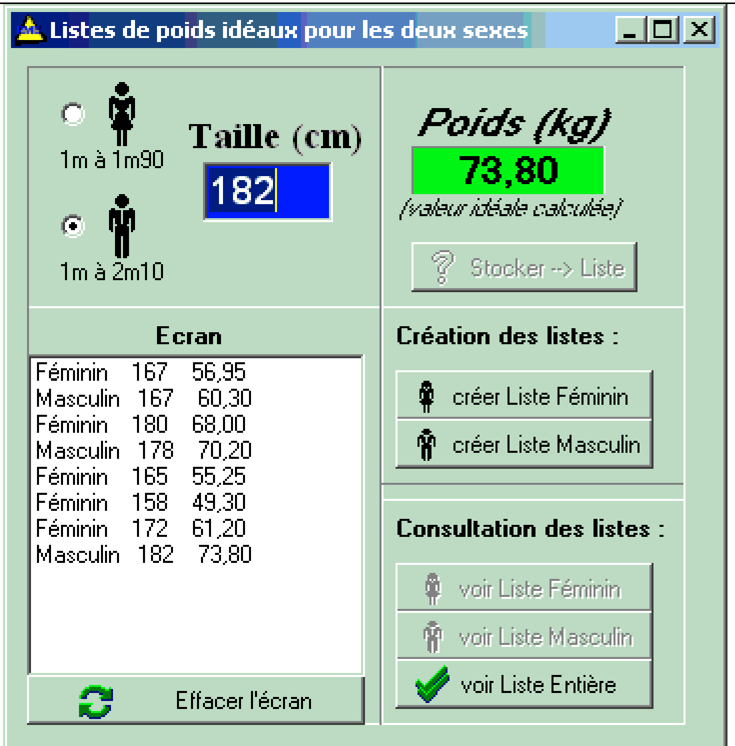
```
end;

{----------------------   PROPRIETE PUBLIC   -------------------------}
function TListe.Getcount:integer;
{fournit le nombre d'éléments dans la propriété Quantite}
begin
 Getcount:=FListGeneric.count
end;

{----------------------   PROPRIETES PUBLISHED   ---------------------}
function TListe.GetDuplic:boolean;
{méthode de lecture de l'autorisation d'élément dupliqué}
begin
 if FListGeneric.duplicates=dupAccept then
  GetDuplic:=true
 else
  getduplic:=false
end;

procedure TListe.SetDuplic(x:boolean);
{méthode d'écriture autorisant ou non la duplication d'éléments}
begin
 if x=true then
  FListGeneric.duplicates:=dupAccept
 else
  flistgeneric.duplicates:=duperror
end;

function TListe.Getsorted:boolean;
{méthode de lecture de l'autorisation de liste triée}
begin
 Getsorted:=FListGeneric.sorted
end;

procedure TListe.Setsorted(x:boolean);
{méthode d'écriture autorisant ou non le tri de la liste}
begin
 if x=true then
  FListGeneric.sorted:=true
 else
  flistgeneric.sorted:=false
end;

end.
```

3.2 Utilisation du composant TListe

Un exemple d'utilisation du composant liste : Il s'agit d'une interface de stockage dans trois listes de type TListe de la taille et du poids idéal d'un individu (à partir d'un exemple d'un ouvrage sur Visual basic) :

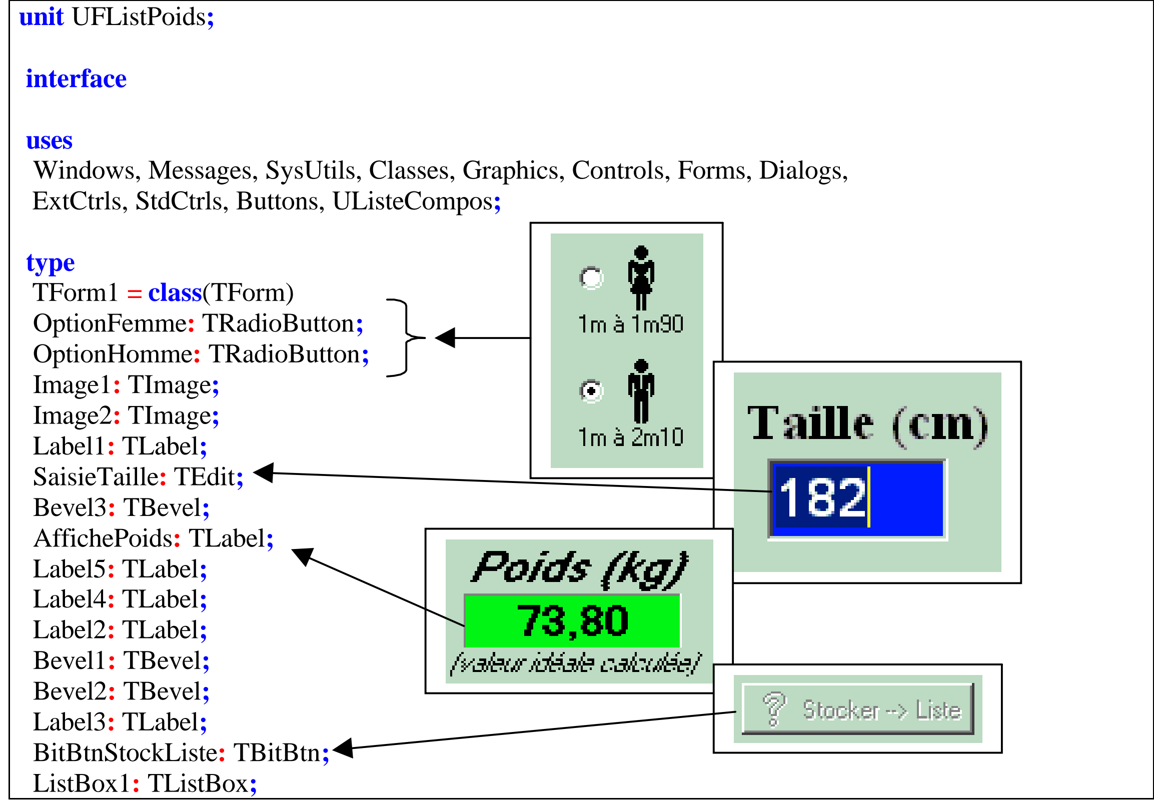
ListeIndiv : TListe
ListeHomme : TListe
ListeFemme : TListe

Au fur et à mesure de la demande de l'utilisateur, le programme range dans la liste " ListeIndiv " les informations sur la personne. Il est possible à chaque instant de construire à partir de cette liste deux sous listes selon le sexe des individus (ListeHomme et ListeFemme).



Le logiciel " PoidsListe" assure un certain niveau de sécurité en utilisant les principes de programmation défensive étudiés au chapitre correspondant.

Code source utilisant le composant TListe




```
unit UFListPoids;

 interface

 uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, Buttons, UListeCompos;

 type
  TForm1 = class(TForm)
  OptionFemme: TRadioButton;
  OptionHomme: TRadioButton;
  Image1: TImage;
  Image2: TImage;
  Label1: TLabel;
  SaisieTaille: TEdit;
  Bevel3: TBevel;
  AffichePoids: TLabel;
  Label5: TLabel;
  Label4: TLabel;
  Label2: TLabel;
  Bevel1: TBevel;
  Bevel2: TBevel;
  Label3: TLabel;
  BitBtnStockListe: TBitBtn;
  ListBox1: TListBox;
```

```
  BitBtnvoirListFemme: TBitBtn;
  BitBtnvoirListHomme: TBitBtn;
  BitBtnListEntiere: TBitBtn;
  Label6: TLabel;
  Label7: TLabel;
  BitBtnCreeListFemme: TBitBtn;
  BitBtnCreeListHomme: TBitBtn;
  Bevel6: TBevel;
  Bevel4: TBevel;
  BitBtnEffaceEcran: TBitBtn;
  Label8: TLabel;
  ListeIndiv: TListe;
  ListeHomme: TListe;
  ListeFemme: TListe;
  Liste1: TListe;
  procedure OptionHommeClick(Sender: TObject);
  procedure OptionFemmeClick(Sender: TObject);
  procedure SaisieTailleChange(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure BitBtnStockListeClick(Sender: TObject);
  procedure BitBtnCreeListFemmeClick(Sender: TObject);
  procedure BitBtnCreeListHommeClick(Sender: TObject);
  procedure BitBtnvoirListFemmeClick(Sender: TObject);
  procedure BitBtnvoirListHommeClick(Sender: TObject);
  procedure BitBtnListEntiereClick(Sender: TObject);
  procedure BitBtnEffaceEcranClick(Sender: TObject);
 private
 { Déclarations privées }
 public
 { Déclarations publiques }
end;
Tsexe=(homme,femme);
Tindividu=class
 public
  Etat:Tsexe;
  Taille:integer;
  Poids:real;
 private
  procedure CalculPoids;
  function CoherenceTaille:Boolean;
end;

 var
  Form1: TForm1;
  Personne:Tindividu;

implementation
const KFemme='Féminin';
 KHomme='Masculin';
var ChaineListe:string;

 {$R *.DFM}
 {--------------------- INITIALISATION DE LA FENETRE -------------------}
 procedure TForm1.FormCreate(Sender: TObject);
 begin
  Personne:=Tindividu.create;
 end;

 {----------------- METHODES DE LA CLASSE Tindividu --------------------}
```

voir Liste Féminin
voir Liste Masculin
voir Liste Entière
créer Liste Féminin
créer Liste Masculin

Composants de Tliste non visuels

```
 procedure Tindividu.CalculPoids;
 {calcule le poids si la taille est dans les limites, puis affiche le
résultat dans affichepoids }
 begin
  if CoherenceTaille then
  begin
   case
    etat of
    femme:
    begin
     Poids:=(Taille-100)*0.85;
     Form1.AffichePoids.Caption:=FloatToStrF(Poids,ffFixed,5,2);
     ChaineListe:=KFemme+'   ';
    end;
    homme:
    begin
     Poids:=(Taille-100)*0.9;
     Form1.AffichePoids.Caption:=FloatToStrF(Poids,ffFixed,5,2);
     ChaineListe:=KHomme+'  '
    end;
   end;{case}
   Form1.BitBtnStockListe.enabled:=true; {active le bouton de stockage dans liste}
  end
  else
  begin
   Form1.BitBtnStockListe.enabled:=false; {désactive le bouton de stockage}
   Form1.AffichePoids.caption:=''; {efface l'affichage du poids}
  end
 end;{-- Fin CalculPoids --}

 function Tindividu.CoherenceTaille:Boolean;
 {indique si la taille est dans les limites acceptées}
 begin
  if (not(Taille in [100..190])and(Etat=Femme))or
   (not(Taille in [100..210])and(Etat=Homme))then
  CoherenceTaille:=false
  else
   coherencetaille:=true
 end; {-- Fin CoherenceTaille --}

 {///////////////////   LES OBJETS VISUELS  ///////////////////////}

 {----------------------- LES OBJETS DE SAISIE -----------------------}
 procedure TForm1.OptionHommeClick(Sender: TObject);
 {permet de saisir le champ etat}
 begin
  Personne.Etat:=homme;
  if SaisieTaille.Text<>'' then
   SaisieTaille.clear;
  SaisieTaille.SetFocus;{par souplesse pour l'utilisateur}
 end;

 procedure TForm1.OptionFemmeClick(Sender: TObject);
 {permet de saisir le champ etat}
 begin
  Personne.Etat:=femme;
  if SaisieTaille.Text<>'' then
   SaisieTaille.clear;
  SaisieTaille.SetFocus;{par souplesse pour l'utilisateur}
 end;
```

```
procedure TForm1.SaisieTailleChange(Sender: TObject);
{permet de saisir le champ taille}
begin
 if OptionFemme.Checked or OptionHomme.Checked then
 begin
  try
   Personne.Taille:=StrtoInt(SaisieTaille.Text);
  except  {le contenu n'est pas entier ? => "EconvertError"}
   on econverterror do
   personne.taille:=0;
  end;
  Personne.CalculPoids
 end
end;

{----------------  STOCKAGE DANS LA LISTE PRINCIPALE  -----------------}

procedure TForm1.BitBtnStockListeClick(Sender: TObject);
begin
 BitBtnStockListe.enabled:=false;
 ListeIndiv.Ajouter(ChaineListe+' '+SaisieTaille.Text+'    '+AffichePoids.caption);
 ListeIndiv.VersListBox(ListBox1); {visualise sur écran la liste}
 SaisieTaille.SetFocus;          {par souplesse pour l'utilisateur}
 BitBtnListEntiere.enabled:=true;
 BitBtnCreeListHomme.enabled:=true; {active le bouton création liste Hom}
 BitBtnCreeListFemme.enabled:=true; {active le bouton création liste Fem}
 BitBtnEffaceEcran.enabled:=true;   {active le bouton effacement d'écran}
 BitBtnvoirListFemme.enabled:=false;{désactive le bouton voir liste Fem}
 BitBtnvoirListHomme.enabled:=false;{désactive le bouton voir liste Hom}
end;

{----------------------- CREATION DES LISTES -----------------------}

procedure ExtraitListe(Clef:string;ListSource:Tliste;Var ListBut:Tliste);
{extrait une sous-liste selon une clef principale}
var i:integer;
begin
 if ListSource.Quantite>0 then {la liste n'est pas vide}
 begin
  ListBut.Effacer;
  for i:=1 to ListSource.Quantite do
   if pos(clef,ListSource.Element(i))<>0 then
    ListBut.Ajouter(ListSource.Element(i))
 end
 else
 begin
  Form1.BitBtnCreeListHomme.enabled:=false; {désactive le bouton création liste Hom}
  Form1.BitBtnCreeListFemme.enabled:=false; {désactive le bouton création liste Fem}
 end
end;

procedure TForm1.BitBtnCreeListFemmeClick(Sender: TObject);
{bouton de création de la liste Fem}
begin
 ExtraitListe(KFemme,ListeIndiv,ListeFemme);
 BitBtnvoirListFemme.enabled:=true;
end;

procedure TForm1.BitBtnCreeListHommeClick(Sender: TObject);
```

```
{bouton de création de la liste Hom}
begin
 ExtraitListe(KHomme,ListeIndiv,ListeHomme);
 BitBtnvoirListHomme.enabled:=true;
end;

{-------------------- VISUALISATION DES LISTES ------------------------}

procedure TForm1.BitBtnvoirListFemmeClick(Sender: TObject);
begin
 ListeFemme.VersListBox(ListBox1); {visualise sur écran la liste}
end;

procedure TForm1.BitBtnvoirListHommeClick(Sender: TObject);
begin
 ListeHomme.VersListBox(ListBox1); {visualise sur écran la liste}
end;

procedure TForm1.BitBtnListEntiereClick(Sender: TObject);
begin
 ListeIndiv.VersListBox(ListBox1); {visualise sur écran la liste}
end;

{------------------------- EFFACEMENT ECRAN -------------------------}

procedure TForm1.BitBtnEffaceEcranClick(Sender: TObject);
begin
 ListBox1.clear;
end;

end.
```

Ici aussi le lecteur est encouragé à modifier et à ajouter de nouvelles fonctionnalités en se servant du source fourni dans l'exemple comme base de travail.

Chapitre 8.2 Les messages Windows

Plan du chapitre:

1. La programmation dirigée par les messages

1.1 Que sont et à quoi servent les messages
1.2 Les messages systèmes
1.3 Delphi et les messages

2. Mécanisme de la répartition des messages en Delphi

2.1 property OnMessage
2.2 procedure MainWndProc
2.3 procedure Dispatch
2.4 Interception directe d'un message
2.5 procedure DefaultHandler
2.6 Gestionnaire d'événement

3. Création et envoi de message en Delphi

3.1 Envoyer un message avec PostMessage et SendMessage
3.2 Exemple d'utilisation

1. La programmation dirigée par les messages

La programmation événementielle peut être présentée comme deux attitudes à adopter lors de la construction d'une application dans un système comme Windows :

> Réagir à des événements (des messages système).
> Engendrer des messages (pour simuler des événements).

Dans les deux cas la notion de messages est présente, ce sont donc des outils de base de ce type de programmation.

1.1 Que sont et à quoi servent les messages

Les messages sont la base de la communication à l'intérieur du système d'exploitation. Ils sont le **résultat d'événements** que le système intercepte ou de **communications du système** avec ses différents composants.

Certaines actions extérieures ou intérieures au système déclenchent des événements. Windows génère alors en continu, un flot de messages en direction des applications ou vers d'autres parties du système lui-même dans le cas de messages internes. Comme son nom l'indique Windows s'intéresse tout particulièrement aux fenêtres, donc pour chaque fenêtre présente, ces messages sont stockés dans une file d'attente (de type FIFO) et sont traités au fur et à mesure par le système.

Certains messages de cette file d'attente sont associés à des événements particuliers. Un tel message peut ainsi être récupéré par une application : nous dirons alors que l'application "**réagit à l'événement**".

Rappellons que si l'on se place soit du point de vue de l'utilisateur, soit de celui d'une application (*ce qui est notre cas*), Windows devient alors semblable à une grande boucle qui attend un événement d'où qu'il vienne, le stocke dans la file des messages, puis le traite. Voici pour une fenêtre le pseudo-comportement de Windows :

```
tantque non ArrêtSysteme faire
  si événement alors
   construire Message ;
   si Message ≠ ArrêtSysteme alors
      reconnaître la fenêtre à qui est destinée ce Message;
      distribuer ce Message
   fsi
  fsi
ftant
```

Dans ce document, la **partie traitement** des messages destinés à un contrôle d'une fenêtre ou d'une application Delphi, est le seul mécanisme qui nous intéresse dans notre programmation événementielle. Nous pouvons utilement et très schématiquement, matérialiser un tel traitement sous la forme de la boucle suivante :

tantque FIFO des messages non vide **faire**
 lire en tête de FIFO le **Message**;
 construire une structure associée à ce **Message**;
 utiliser la méthode interne de traitement sur ce **Message**
ftant

Remarque

> Pendant que ce traitement a lieu, Windows continue en outre à intercepter d'autres événements et à stocker dans la file d'attente les messages associés à ces événements et les messages internes.

En résumé nous pouvons énoncer l'affirmation suivante :

Le système **envoie** ou **poste** des messages prédéfinis à une application, mais réciproquement une application peut **envoyer** ou **poster** des messages vers d'autres fenêtres ou d'autres applications.

1.2 Les messages systèmes dans Windows

Chaque message système dispose d'un identificateur unique du message. Il correspond à une **constante numérique** de base de Windows. Chaque identificateur de message est associé à une action spécifique.

Le classement des messages dans Windows s'effectue par un préfixe qui permet de déterminer la catégorie à laquelle appartient le message.

Soit par exemple le message dont l'identificateur est **WM_KEYDOWN**, il appartient à la catégorie des messages généraux de Windows **WM_**xxx. C'est un message associé à l'appui d'une touche de clavier. Le préfixe de l'identificateur indique le type de fenêtre qui peut intercepter et donc réagir à ce message.

> Les messages **WM_**xxx sont des **W**indow **M**essages donc destinés à toutes les fenêtres.
> Les messages **LB_**xxx sont des **L**ist **B**ox messages destinés aux boîtes de listes.
> Les messages **EM_**xxx sont des **E**dit **M**essages destinés aux contrôles d'édition.
> Les messages **SBM_**xxx sont des **S**croll **B**arre **M**essages destinés aux barres de défilement.
> etc...

L'aide en ligne de l'API W32 de Windows nous fournit à toutes fins utiles, la structure générale d'un message de Windows :

voici cette structure en C :	voici cette structure en Delphi :
typedef struct tagMSG { *// msg* HWND **hwnd**; UINT **message**; WPARAM **wParam**; LPARAM **lParam**; DWORD **time**; POINT **pt**; } MSG;	**type** TMsg = **packed record** **hwnd**: HWND; **message**: UINT; **wParam**: WPARAM; **lParam**: LPARAM; **time**: DWORD; **pt**: TPoint; **end**;

Ci-dessous la signification des champs de la structure d'un message :

hwnd = la référence de la fenêtre à qui est destinée le message.
message = la **constante numérique** identifiant le message.
wParam = entier positif codé sur quatre octets [0..4294967295] représentant un premier paramètre dont la signification dépend de la catégorie du message.
lParam = entier quelconque codé sur quatre octets [–2147483648..2147483647] représentant un deuxième paramètre dont la signification dépend de la catégorie du message.
time = heure système à laquelle le message a été créé par Windows.
pt = position du curseur de souris en coordonnées d'écran (pt.x, pt.y :Longint)

Exemple:

Lors de l'appui sur une touche clavier que va-t-il se passer ?

Si vous tapez sur la touche "M" du clavier, une série de messages est envoyée en cascade :

- Un message **WM_KEYDOWN** est automatiquement envoyé par Windows dans la file d'attente de la fenêtre détenant la focalisation (nommons la **FenWin**), dès que la touche est enfoncée, et ceci tant que la touche est enfoncée.
- Ensuite Windows envoie le message **WM_CHAR** (qui contient le caractère entré au clavier) dans la même file;
- puis lorsque vous relâchez la touche, il envoie enfin un message **WM_KEYUP**.

Que contient par exemple, la structure **TMsg** lors du traitement de ce message **WM_KEYDOWN** *(information obtenue à partir de l'aide en ligne de l'API Win32)*:

hwnd: référence (pointeur,adresse) de la fenêtre **FenWin**.
message: **WM_KEYDOWN**,(valeur numérique=256)
wParam: = 77 (code virtuel de la touche M ici)
lParam: les 32 bits de ce mot indiquent les informations suivantes :

- bits 0..15 : le nombre sur 16 bit indique combien de fois il faut répéter le caractère. Il correspond au maintien de la touche enfoncée.

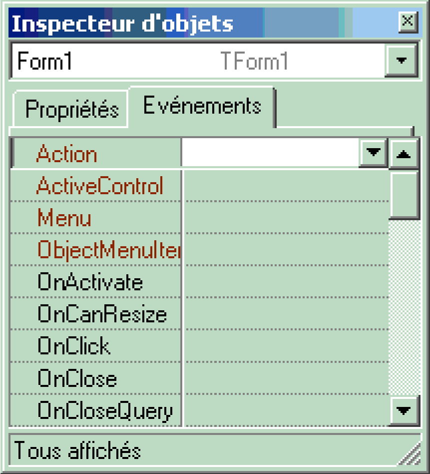
- bits 16..23 : un code spécifique au clavier (OEM)
- bit 24 : si bit = 1 c'est une touche étendue (ALT+,CTRL+,...) sinon bit = 0
- bit 25..28 : réservé par le système.
- bit 29 : bit de contexte = 0 pour ce message.
- bit 30 : bit d'état précédent de cette touche si bit = 1 la touche est déjà enfoncée, sinon la touche était relevée et alors bit = 0
- bit 31 : bit d'état de transition = 0 pour ce message.

1.3 Delphi et les messages

Une application Delphi est basée sur une fenêtre d'application. Nommons notre application **FenDelphi;** elle est donc considérée par Windows comme une fenêtre quelconque et à ce titre elle peut recevoir et envoyer des messages.

Une application Delphi possède des outils d'interception de la structure **TMsg**, de son décodage et de son interprétation. Nous avons déjà vu que Delphi disposait, à l'attention du programmeur, pour chaque objet et pour certains événements, de **gestionnaires d'événement.**

Ces gestionnaires sont des outils de premier niveau (en fait les plus abstraits et les plus encapsulés)de gestion des messages. L'inspecteur d'objet nous a permis de nous familiariser avec de tels gestionnaires dans son onglet événements.



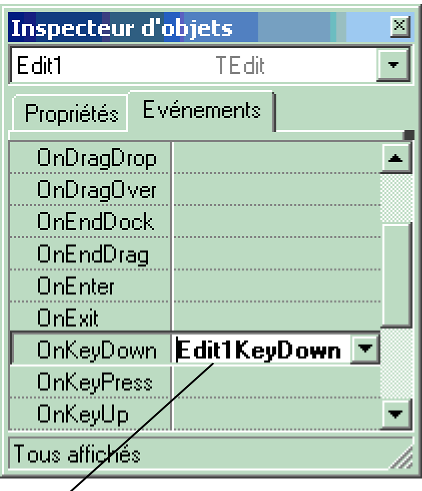
Onglet - événements

Si nous reprenons l'exemple précédent de l'appui sur la touche "M" du clavier lorsque **FenDelphi** détient la focalisation, nous savons que Windows va construire la structure **TMsg** précédente et l'ajouter dans la file d'attente des messages de **FenDelphi**.

Supposons que dans notre application **FenDelphi** nous ayons déposé sur la fiche **Form1** deux contrôles visuels **Edit1** et **Edit2** de la classe des **TEdit,** et que ce soit **Edit1** qui détienne le focus lors du lancement de l'application **FenDelphi** :

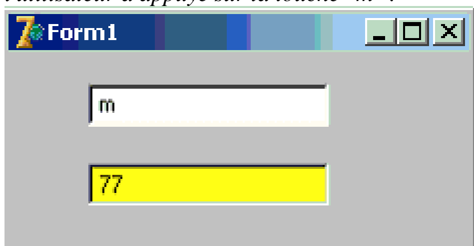
Edit détient le focus dans Form1 de ***FenDelphi***

Nous souhaitons faire réagir **Edit1** à l'appui sur la touche "M" du clavier de la façon suivante : dès que l'utilisateur appuie sur une touche du clavier dans **Edit1**, il apparaît dans **Edit2** le code de ce caractère.
Nous allons utiliser le gestionnaire de l'événement OnKeyDown de **Edit1** :




```
procedureTForm1.Edit1KeyDown(Sender: TObject; var Key: Word;  Shift: TShiftState);
begin
  Edit2.text:=inttostr(Key) //code du caractère entré dans Edit, recopié dans Edit2
end;
```

l'utilisateur a appuyé sur la touche "m" :



Voici l'aide en ligne de DELPHI sur l'événement **OnKeyDown** :

```
type
  TKeyEvent = procedure (Sender: TObject; var Key: Word; Shift: TShiftState) of object;
property OnKeyDown: TKeyEvent;
```

- Le gestionnaire d'événement **OnKeyDown** permet d'effectuer un traitement spécifique quand une touche est enfoncée. Le gestionnaire **OnKeyDown** peut répondre à toutes

les touches du clavier, y compris les touches de fonction et les combinaisons avec les touches Maj, Alt et Ctrl ainsi qu'avec les boutons de la souris.

- Le paramètre Key indique la touche du clavier.
- Le type TKeyEvent pointe sur une méthode (un gestionnaire d'événement) gérant les événements du clavier : ici TForm1.Edit1KeyDown
- **OnKeyDown** est déclenché automatiquement lorsque le message **WM_KEYDOWN** est intercepté par l'application.

Il existe donc un mécanisme interne à Delphi qui a permis à notre application de reconnaître que Windows avait envoyé le message **WM_KEYDOWN** dans la structure **TMsg**, ce mécanisme a permis aussi d'appeler le gestionnaire **TForm1.Edit1KeyDown** que nous avons écrit afin d'effectuer la recopie du code dans Edit2.

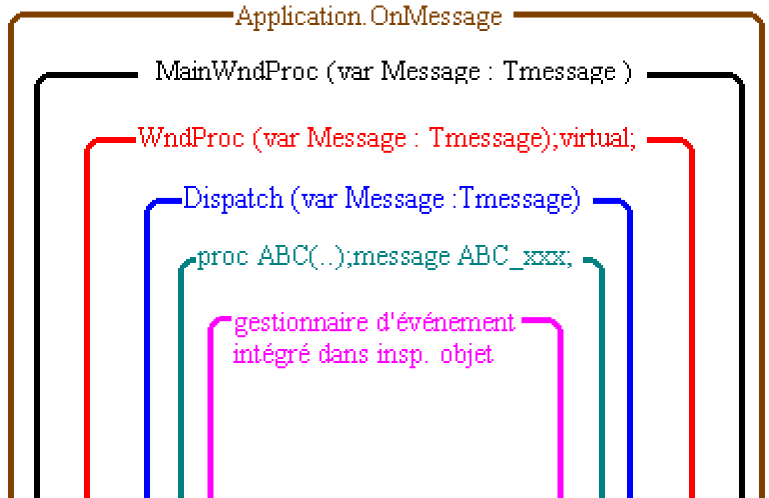
2. Mécanisme de répartition des messages en Delphi

Delphi répartit les messages de plusieurs façons :

Chaque composant hérite d'un système complet de répartition de message. Toutes les classes Delphi disposent d'un mécanisme intégré pour gérer les messages : ce sont les méthodes de gestion des messages ou **gestionnaires de messages,** ces méthodes sont choisies dans un ensemble de méthodes spécifiques.

Le système de répartition de message dispose d'une gestion par défaut. Vous ne définissez de gestionnaire que pour les messages auxquels vous souhaitez spécifiquement répondre. Un gestionnaire par défaut est appelé si aucune méthode n'est définie pour le message.

Le diagramme suivant illustre une partie essentielle du fonctionnement du système de répartition de message dans Delphi et les différents niveaux d'interception possibles d'un message :



property OnMessage: TMessageEvent;
procedure MainWndProc(**var** Message: TMessage);
procedure WndProc(**var** Message: TMessage); **virtual**;

```
procedure Dispatch(var Message); virtual;
procedure ABCxxx(var Message: TMessage);message ABC_xxx;
procedure DefaultHandler(var Message); virtual;
```

Procédons à l'examen de chaque niveau d'interception

2.1 niveau : property OnMessage

Si vous voulez intercepter tout ou partie des messages Windows expédiés à toutes les fenêtres de l'application Delphi, utilisez l'événement OnMessage de la classe **TApplication** qui encapsule toute application Delphi. Cet événement OnMessage est déclenché dès que votre application reçoit un quelconque message de Windows.

```
property OnMessage: TMessageEvent;
type TMessageEvent = procedure (var Msg: TMsg; var Handled: Boolean) of object;
```

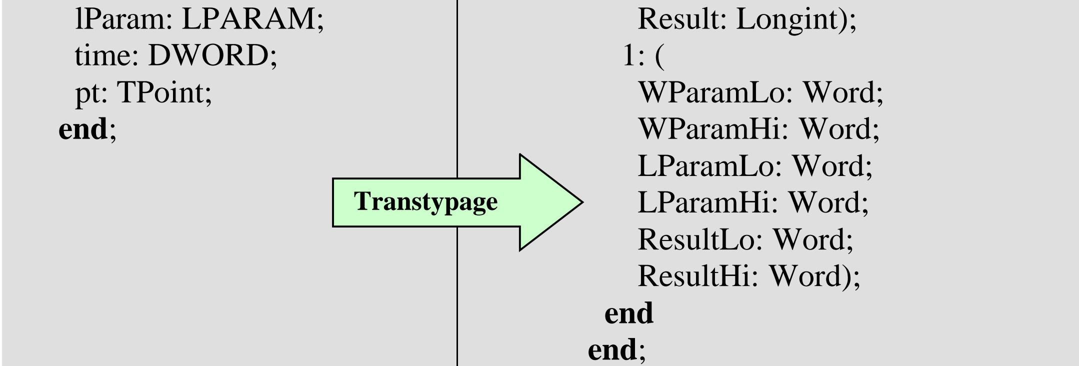
L'écriture du gestionnaire d'événement OnMessage vous permet de répondre à tous les messages que reçoit l'application:

```
type
 TForm1 = class(TForm)
   ........
   private
   procedure MonAppliMessages(var Msg: TMsg; var Handled: Boolean);
 end;

implementation

procedure TForm1.MonAppliMessages(var Msg: TMsg; var Handled: Boolean);
begin
 //traitement du message contenu dans Msg du type TMsg
  if Msg.message=WM_LBUTTONDOWN then ....Traitement
end;
//Le gestionnaire étant créé par l'instruction Application.OnMessage:=MonAppliMessages :
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMessage := MonAppliMessages;
end;
```

si vous ne créez pas un tel gestionnaire le message est distribué à la fenêtre à laquelle il est destiné et Delphi continue la suite de tentative de traitement du message pour la fenêtre concernée. Tout d'abord Delphi transtype la struture TMsg en une structure interne de type TMessage :

structure Tmsg en Delphi :	Transtypage →	structure TMessage en Delphi :

<pre>type
  TMsg = packed record
    hwnd: HWND;
    message: UINT;
    wParam: WPARAM;</pre>
		<pre>type
		  TMessage = record
		    Msg: Cardinal;
		    case Integer of
		      0: (
		        WParam: LongInt;
		        LParam: Longint;</pre>



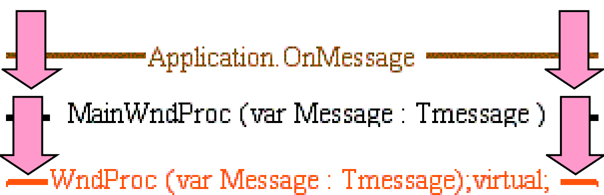
Le champ hwnd n'a plus de raison d'exister puisque la fenêtre à qui le message s'adresse a déjà été identifiée. Les champs time et pt de la structure **TMsg** sont réintégrés à l'intérieur d'autres champs de la structure **TMessage**.

Remarque

> OnMessage **ne reçoit que des messages envoyés à la file d'attente Fifo des messages**, et non ceux envoyés directement (en particulier ceux de SendMessage).

2.2 niveau : procedure MainWndProc - WndProc

L'interception ayant eu lieu la redirection vers la fenêtre adéquate est assurée par la **procedure** MainWndProc(**var** Message: TMessage); c'est la **procedure** WndProc(**var** Message: TMessage); **virtual**; surchargeable, qui va se charger de traiter ou transmettre le message.



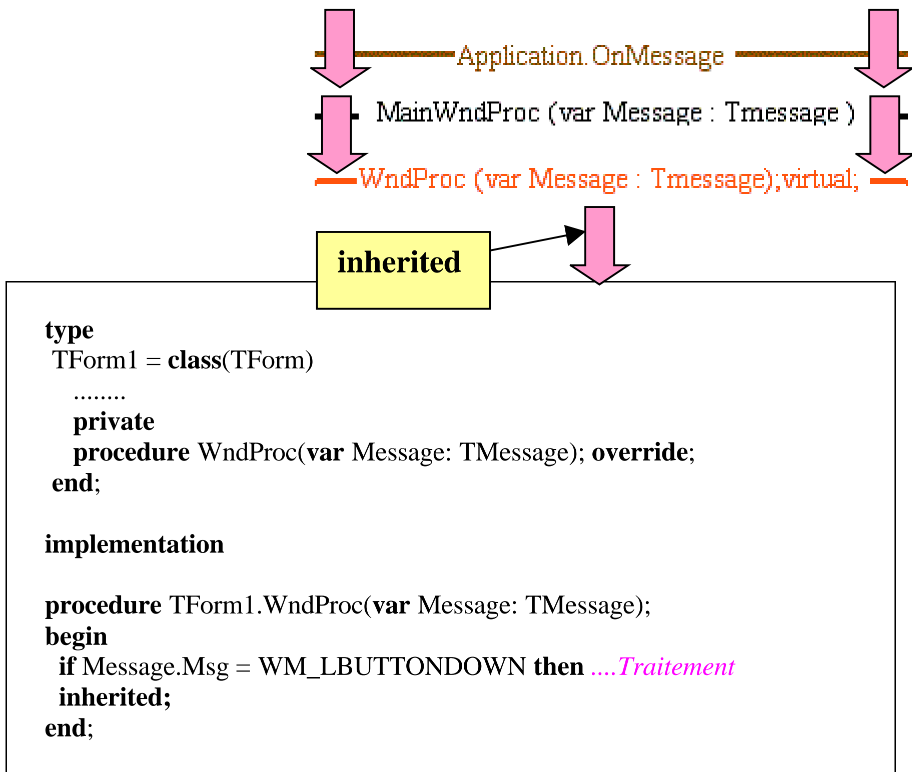
Exemple :
soit dans une application à intercepter l'enfoncement du bouton gauche de la souris.

```
type
 TForm1 = class(TForm)
   ........
   private
   procedure WndProc (var Message: TMessage); override;
 end;

implementation

procedure TForm1.WndProc(var Message: TMessage);
begin
   if Message.Msg = WM_LBUTTONDOWN then ....Traitement
end;
```

Le traitement ci-dessus de WM_LBUTTONDOWN dans WndProc arrête la diffusion des messages vers les autres niveaux d'encapsulation.

Le code de la procédure WndProc tel qu'il est écrit ci-dessus est inefficace et peut produire un message d'erreur car de tous les messages passant par WndProc nous n'avons décidé de n'en traiter qu'un seul le : WM_LBUTTONDOWN, en oubliant les autres messages de la fenêtre comme le positionnement, l'affichage etc...

Pour laisser Delphi s'occuper du reste il suffit de rajouter **inherited** à la fin de WndProc et la transmission des autres messages pourra se poursuivre.




```
type
 TForm1 = class(TForm)
   ........
   private
   procedure WndProc(var Message: TMessage); override;
 end;

implementation

procedure TForm1.WndProc(var Message: TMessage);
begin
  if Message.Msg = WM_LBUTTONDOWN then ....Traitement
  inherited;
end;
```

2.3 niveau : procedure Dispatch

La méthode WndProc reçoit tous les messages destinés à la fenêtre et fait un filtrage. Elle appelle la méthode Dispatch pour chaque message selectionné, cette méthode n'a pas de type au message qui est transmis, c'est un message générique. Si vous voulez intercepter un message spécifique à ce niveau vous devrez le transtyper par exemple en TMessage.

```
type
 TForm1 = class(TForm)
   ........
   private
   procedure Dispatch(var Message); override;
 end;

implementation

procedure TForm1.Dispatch(var Message);
begin
```

WndProc (var Message : Tmessage);virtual;
Dispatch (var Message :Tmessage)

```
  if Tmessage (Message).Msg = WM_LBUTTONDOWN then ....Traitement
 end;
```

2.4 niveau : Interception de redéfinition d'un message

procedure ABCxxx(**var** Message: TMessage);**message** ABC_xxx;

Ce mécanisme direct vous autorise à créer une méthode spécifique en redéfinissant le traitement d'un message donné du système Windows. Il s'agit d'une redéfinition, car le message ABC_xxx est traité par la classe, la **procedure** ABCxxx(**var** Message: TMessage) remplace ce traitement habituel (liaison statique) par le vôtre. L'utilisation d'inherited se justifie pleinement si l'on souhaite laisser le traitement habituel avoir lieu.

En reprenant le même exemple d'enfoncement du bouton gauche de souris :

```
type
 TForm1 = class(TForm)
   ...........
   private
     procedure WMLBUTTONDOWN(var Message:Tmessage);message WM_LBUTTONDOWN;
 end;

implementation

procedure TForm1.WMLBUTTONDOWN(var Message:Tmessage);
begin
  ....Traitement
 {si vous mettez inherited et qu'un gestionnaire de l'événement a été écrit par le  programmeur pour
un événement déclenché par WM_LBUTTONDOWN, vous pouvez laisser le message continuer à être
traité.}
end;
```

Au chapitre précédent dans le composant TwinArbre nous construit une telle procédure :

```
private
 procedure WMSize(var Message:TWMsize); message WM_SIZE;
 ......
procedure TwinArbre.WMSize(var Message:TWMsize);
{permet de modifier la taille du TCustomcontrol lors de la conception}
begin
   inherited;
   Ftree.setbounds(0,0,width,height-25);
   FPotentiometre.setbounds(0,FTree.Top+FTree.Height,width-25,25);
   FEdit1.setbounds(FPotentiometre.left+FPotentiometre.width, Ftree.Top+Ftree.Height,25,25);
end;
```

Elle intercepte et redéfini le message WM_SIZE, qui est envoyé au composant TCustomControl dès qu'une de ses dimensions est modifiée. Nous avons mis inherited afin de laisser se propager le message WM_SIZE aux niveaux en-dessous et afin que le TCustomControl réagisse correctement , puis nous avons programmé le redimensionnement de chacun des composants visuels déposé sur le TcustomControl.

2.5 niveau : procedure DefaultHandler

Si aucun gestionnaire n'a été écrit pour ce message, le système de gestion par défaut de Delphi s'en charge à votre place : il appelle la méthode **DefaultHandler**. Comme pour Dispatch le paramètre est de type message générique. Il s'agit ici d'un mécanisme qui se trouve juste au dessus de celui du gestionnaire d'événement.
DefaultHandler permet en particulier de gérer par surcharge tous les messages pour lesquels l'objet n'a pas de gestionnaire spécifique ou bien d'intercepter (c'est le denier niveau possible) un message connu par l'objet.

```
type
 TForm1 = class(TForm)
   ...........
   private
     procedure DefaultHandler(var Message); override;
 end;

implementation

procedure TForm1.DefaultHandler(var Message);
begin
  if  TMessage(Message).Msg = WM_LBUTTONDOWN then ....Traitement
end;
```

2.6 niveau : Gestionnaire d'événement

Enfin si un gestionnaire d'événement a été écrit par le programmeur, la méthode Dispatch l'appelle. Il s'agit des gestionnaires d'événements qui sont fournis par Delphi avec une entrée dans l'inspecteur d'objet, ici pour l'interception de l'enfoncement de la souris l'événement OnMouseDown est géré. L'événement **OnMouseDown** est déclenché automatiquement lorsque l'un des messages suivants **WM_LBUTTONDOWN**, **WM_MBUTTONDOWN**, **WM_RBUTTONDOWN** est intercepté par l'application :

```
type
 TForm1 = class(TForm)
   procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
     Shift: TShiftState; X, Y: Integer);
   private
     ........
 end;

implementation

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
     Shift: TShiftState; X, Y: Integer);
begin
   ....Traitement
end;
```

Exemple : affichage de la hiérarchie des niveaux d'interception du ***WM_LBUTTONDOWN***

Dans cet exemple nous interceptons le click gauche de souris à tous les niveaux, et nous laissons l'interception continuer aux niveaux inférieurs grâce au **inherited**. Ci-dessous le code source de la unit du programme Delphi correspondant :

```
Unit Unitexemple;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Memo1: TMemo;
    Label1: TLabel;
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
  private
    { Déclarations privées }
    procedure MonAppliMessages(var Msg: TMsg; var Handled: Boolean);
    procedure WndProc(var Message: TMessage); override;
    procedure Dispatch(var Message);override;
    procedure DefaultHandler(var Message); override;
    procedure WMLBUTTONDOWN(var Message:Tmessage);message WM_LBUTTONDOWN;
  public
    { Déclarations publiques }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
 Application.OnMessage:= MonAppliMessages;
end;
procedure TForm1.MonAppliMessages(var Msg: TMsg; var Handled: Boolean);
begin
  if Msg.message=WM_LBUTTONDOWN then
  begin
   Edit1.text:=inttostr(Msg.wParam);
   Edit2.text:=inttostr(Msg.lParam);
   Edit3.text:=inttostr(Msg.time);
   Edit3.color:=clBlue;
   Memo1.Lines.Add('intercepté : niveau 1 - MonAppliMessages')
  end;
  inherited
end;
procedure TForm1.WndProc(var Message: TMessage);
begin
 if Message.Msg = WM_LBUTTONDOWN then
 begin
  Edit1.text:=inttostr(Message.WParam);
  Edit2.text:=inttostr(Message.LParamHi) + '/' + inttostr(Message.LParamLo);
  Edit3.text:=inttostr(Message.Result);
  Edit3.color:=clred;
```

```
 Memo1.Lines.Add('intercepté : niveau 1 - WndProc')
 end;
 inherited;
end;
procedure TForm1.Dispatch(var Message);
begin
 if  Tmessage(Message).msg = WM_LBUTTONDOWN then
 begin
  Edit1.text:=inttostr(Tmessage(Message).WParam);
  Edit2.text:=inttostr(Tmessage(Message).LParamHi) + '/' + inttostr(Tmessage(Message).LParamLo);
  Edit3.text:=inttostr(Tmessage(Message).Result);
  Edit3.color:=clyellow;
  Memo1.Lines.Add('intercepté : niveau 2 - Dispatch')
 end;
 inherited;
end;
procedure TForm1.WMLBUTTONDOWN(var Message:Tmessage);
begin
  Edit1.text:=inttostr(Message.WParam);
  Edit2.text:=inttostr(Message.LParamHi) + '/' + inttostr(Message.LParamLo);
  Edit3.text:=inttostr(Message.Result);
  Edit3.color:=cllime;
  Memo1.Lines.Add('intercepté : niveau 3 - Proc WMLBut') ;
  inherited;
end;
procedure TForm1.DefaultHandler(var Message);
begin
if Tmessage(Message).msg = WM_LBUTTONDOWN then
 begin
  Edit1.text:=inttostr(Tmessage(Message).WParam);
  Edit2.text:=inttostr(Tmessage(Message).LParamHi)+'/'+inttostr(Tmessage(Message).LParamLo);
  Edit3.text:=inttostr(Tmessage(Message).Result);
  Edit3.color:=clAqua;
  Memo1.Lines.Add('intercepté : niveau 4 - DefaultHandler')
 end;
 inherited;
end;
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
 Edit3.color:=cllime;
 Memo1.Lines.Add('intercepté : niveau 5 - gest. OnMouseDown') ;
end;
end.
```

Si l'on click sur le fond de la fiche voici l'ordre d'interception en cascade du message WM_LBUTTONDOWN :

Remarquons que l'Edit3 est en couleur clLime, ce qui est normal puisque c'est le niveau 6 du gestionnaire d'événement OnMouseDown qui est exécuté en dernier.

Si l'on click dans n'importe lequel des contrôles déposés sur la fiche (un des Edit, ou le memo) voici ce que nous obtenons :

C'est le gestionnaire MonAppliMessages de l'événement OnMessage de l'application qui a été le seul à être exécuté. Ce qui signifie que l'application a bien reçu de la part du système le message WM_LBUTTONDOWN et que son gestionnaire MonAppliMessages l'a pris en compte. Toutefois comme nous avions clické dans l'Edit2 (Edit jaune) pour la suite Delphi n'ayant aucun traitement proposé par le programmeur, a abandonné le traitement de ce message.

3. Création et envoi de message en Delphi

Il est possible de travailler avec des messages définis par l'utilisateur. Une application Delphi traitera ces messages comme elle traite les autres messages système.

Vous devez déclarer un identificateur de message personnel. Un identificateur de message est une constante entière numérique. Windows se réserve pour son propre usage les messages dont le numéro est inférieur à 1024. Lorsque vous déclarez vos propres messages, vous devez donc toujours débuter par un numéro supérieur ou égal à 1024.

Delphi vous fournit si vous souhaitez l'utiliser, un identificateur de constante **WM_USER** représentant le numéro de départ (WM_USER = 1024) pour vos messages.

Voici trois constantes de messages personnels :

```
const
 MonMessage1 = WM_USER + 100;
 MonMessage2 = WM_USER + 120;
 MonMessage3 = 1500;
```

3.1 Envoyer un message avec PostMessage et SendMessage

L'API Windows met à notre disposition de nombreuses procédures de bas niveau permettant d'envoyer un message à un contrôle fenêtré (descendant des TWinControl), dont les plus simples sont **PostMessage** et **SendMessage**. Il est rare que nous soyons obligés d'utiliser ce genre de fonction de bas niveau, mais ici l'objectif est d'indiquer comment envoyer des messages personnalisés.

SendMessage (procédure de traitement synchrone) attend que le message soit traité, si le message est envoyé à un autre thread, l'application émettrice du message reste bloquée tant que l'application receptrice n'a pas fini de le traiter.
en-tête de la fonction :

function SendMessage(hwnd : HWND; Msg : cardinal; wParam , lParam : Longint) : Longint;

Remarque

SendMessage envoie directement le message à une fenêtre **sans passer par la file d'attente**. L'événement OnMessage de l'application n'est donc pas déclenché, il est conseillé de n'utiliser **SendMessage** qu'à l'intérieur d'une même application pour envoyer des messages synchrones à des contrôles fenêtrés de l'application (Fiche, Panel, Edit, etc...). Dans le cas où vous utilisez **SendMessage pour communiquer avec une autre application,** pensez bien à construire dans l'application réceptrice, une procédure d'interception directe du message personnalisé, ce qui sera le seul moyen d'intercepter ce message.

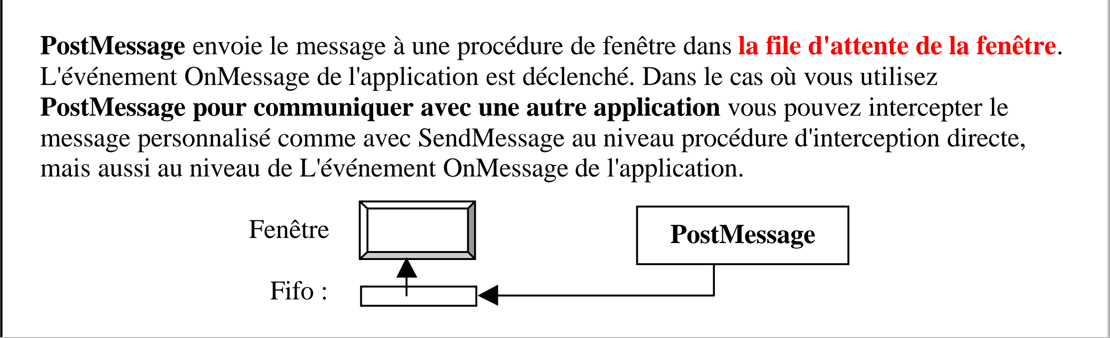
Fenêtre ◄── **SendMessage**

Fifo :

PostMessage (procédure de traitement asynchrone)envoie le message dans la file d'attente des messages du contrôle fenêtré et n'attend pas que le message soit traité.
en-tête de la fonction :

function PostMessage(hwnd : HWND; Msg : cardinal; wParam , lParam : Longint) : Longint;

Remarque


PostMessage envoie le message à une procédure de fenêtre dans **la file d'attente de la fenêtre**. L'événement OnMessage de l'application est déclenché. Dans le cas où vous utilisez **PostMessage pour communiquer avec une autre application** vous pouvez intercepter le message personnalisé comme avec SendMessage au niveau procédure d'interception directe, mais aussi au niveau de L'événement OnMessage de l'application.

Bien entendu, outre vos messages personnalisés, ces deux procédures permettent aussi d'envoyer des messages définis de Windows (WM_xxx, LB_xxx, EM_xxx, etc...)

```
SendMessage(Form1.Handle,WM_CLOSE,0,0); //message de fermeture de la fiche Form1
SendMessage(Form1.Edit1.Handle,WM_CHAR,ord('M'),0); //envoi du caractère 'M' dans Edit1
```

etc...

Ces deux fonctions de bas niveau, ont comme premier paramètre le Handle de fenêtre du contrôle à qui est envoyé le message, il existe pour les contrôles en général, une méthode notée **Perform** qui permet d'envoyer directement un message à la procédure de fenêtre du contrôle sans avoir à connaître son Handle : le contrôle répond alors comme s'il avait reçu le message Windows spécifié.

```
function Perform(Msg: Cardinal; WParam, LParam: Longint): Longint;
```

Enoncé :

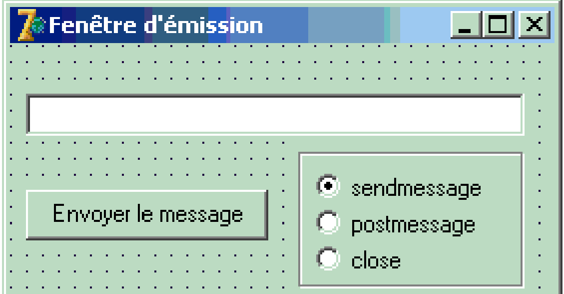
Afficher le même message WM_USER expédié soit par PostMessage, soit par SendMessage, soit fermer la fenêtre principale d'une application PReception.exe, *à partir d'une autre application* PEmissPost.exe, *par d'envoi de messages.*

Application émettrice : **PEmissPost**

Application réceptrice : **Preception**

1 - Source de l'application PEmissPost.exe

Cette application envoie le message WM_USER (si l'un des deux radio-bouton sendmessage ou postmessage est coché), si c'est le radio bouton close qui est coché elle envoie le message WM_USER+1 :



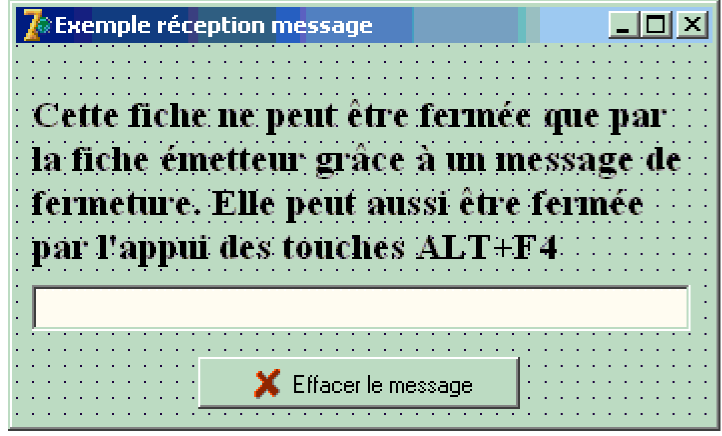
code source complet de l'application

```
unit UFemissPost;

 interface

 uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls;

 type
  TForm1 = class(TForm)
  Button1: TButton;
  EditInfos: TEdit;
  RadioGroupTypeMessage: TRadioGroup;
  procedure Button1Click(Sender: TObject);
  procedure RadioGroupTypeMessageClick(Sender: TObject);
 private
```

```
 { Déclarations privées }
 public
 { Déclarations publiques }
  procedure EnvoyerUnMessage;
end;

 var  Form1: TForm1;

implementation

 {$R *.dfm}
 {------------ Emission  message fermeture à la fenêtre TFExemple ----(émission)---------}
 // fermeture de la fenêtre TFxxxx déjà ouverte avant fermeture
 procedure TForm1.EnvoyerUnMessage;
 var
  Wnd:HWnd;
  begin
   Wnd:=FindWindow('TFExemple',nil); // recherche de la fenêtre TFExemple
   if Wnd<>0 then  // si instance de la fenêtre  trouvée => on lui envoi un message
   begin
    EditInfos.Text:='Fenêtre TFExemple trouvée, message envoyé !';
    case
     radiogrouptypemessage.itemindex of
     0: SendMessage(Wnd,WM_USER,0,0); // message à traiter directement
     1: PostMessage(Wnd,WM_USER,0,0); // message posté dans la fifo
     2: PostMessage(Wnd,WM_USER+1,0,0); // message close posté dans la fifo
    end;
   end
   else
    EditInfos.Text:='Fenêtre TFExemple non trouvée';
  end;

 procedure TForm1.Button1Click(Sender: TObject);
 begin
  EnvoyerUnMessage;
 end;

 procedure TForm1.RadioGroupTypeMessageClick(Sender: TObject);
 begin
  EditInfos.Text:=''
 end;

end.
```

2 - Source de l'application PReception.exe

Cette application reçoit des messages en particulier ceux provenant de l'application précédente lorsque les deux applications sont exécutées en même temps, nous avons programmé la reception et l'interception des messages WM_USER et WM_USER+1 à trois niveaux : 1°)niveau OnMessage, 2°) niveau WndProc , 3°) niveau redéfinition de message.



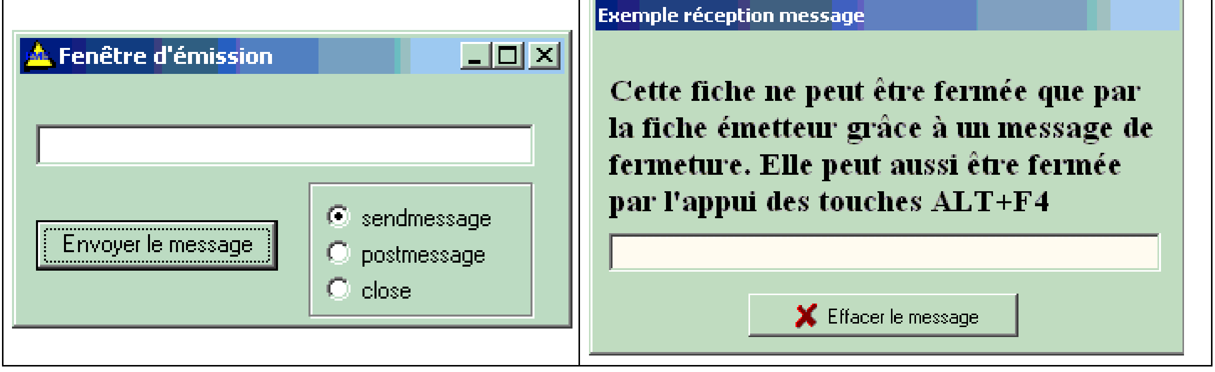
code source complet de l'application

```
unit UFReception;

 interface

 uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Buttons;

 type
  TFExemple = class(TForm)
  Label1: TLabel;
  Edit1: TEdit;
  BitBtnEffacer: TBitBtn;
  procedure FormCreate(Sender: TObject);
  procedure BitBtnEffacerClick(Sender: TObject);
 private
 { interception à 3 niveaux différents}
  procedure MessageUser(var mess:TMsg;var hand:boolean);
  procedure WndProc(var Message: TMessage);override;
  procedure MessWMUSER (var Mess:TMessage);message WM_USER;
 public
end;

 var
  FExemple: TFExemple;

implementation

 {$R *.dfm}
 {--------------- Interception message utilisateur ----(reception)--------------}
 procedure TFExemple.MessageUser(var mess:TMsg;var hand:boolean);
 { traite le message wm_User intercepté comme un ordre donné à la fenêtre.
 Méthode personnelle de traitement des messages reçus de la part d'une autre application }
 begin
  if mess.message=WM_USER then // message envoyé = WM_USER
  begin
   Edit1.Text:='WM_User OnMessage + ';
  end
  else
   if mess.message=WM_USER+1 then // message envoyé = WM_USER+1 (<=>close ici)
   begin
      close  // message reçu interprété ici comme une fermeture "close"
   end;
  {else if mess.message=WM_USER+2 then .....}
```

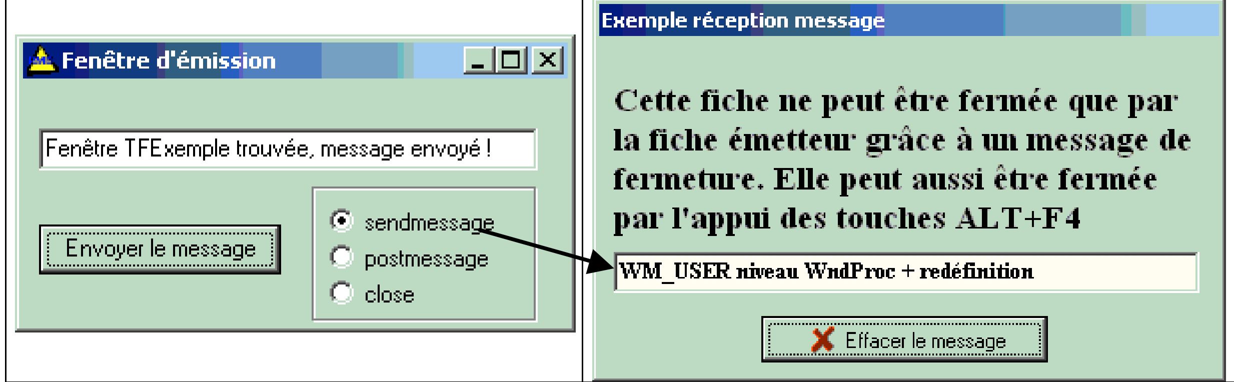
Application.OnMessage
MainWndProc (var Message : Tmessage)
WndProc (var Message : Tmessage);virtual;
Dispatch (var Message :Tmessage)
proc ABC(..);message ABC_xxx;

```
 inherited;
end;

procedure TFExemple.WndProc(var Message: TMessage);
begin
 if Message.msg=WM_USER then // message envoyé = WM_USER (<=>close ici)
 begin
  if Edit1.Text='' then
     Edit1.Text:='WM_USER ';
  Edit1.Text:=Edit1.Text+'niveau WndProc';
 end;
 inherited
end;

procedure TFExemple.MessWMUSER (var Mess:TMessage);
begin
 Edit1.Text:=Edit1.Text+' + redéfinition';
end;

procedure TFExemple.FormCreate(Sender: TObject);
begin
 Application.OnMessage:=MessageUser; //autorise l'interception des messages expédiés à cette fenêtre
end;

procedure TFExemple.BitBtnEffacerClick(Sender: TObject);
begin
 Edit1.Text:=''
end;

end.
```

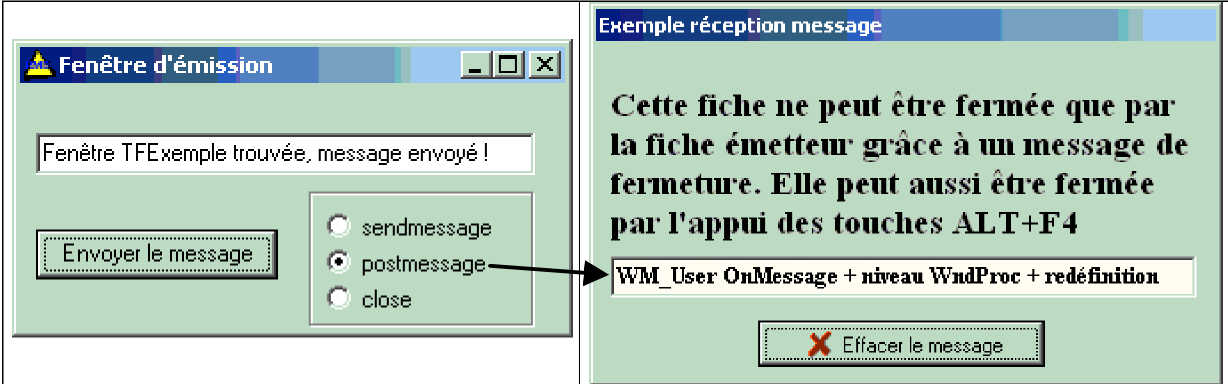
On lance les deux applications **PReception.exe** et **PEmissPost.exe** , voici l'état initial des deux fenêtres de chaque application lancée en même temps :



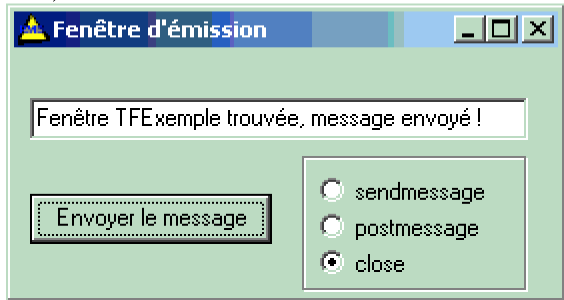
En cliquant sur le bouton"Envoyer le message" comme le radio-bouton sendmessage est sélectionné, c'est l'instruction **"SendMessage(Wnd,WM_USER,0,0);"**qui est exécutée (rappelons qu'elle ne met rien dans la fifo de la fenêtre et que donc OnMessage ne l'intercepte pas, ce que l'on vérifie dans les images d'exécution ci-après :



Si l'on sélectionne le radio-bouton postmessage et que l'on clique sur le bouton"Envoyer le message", c'est l'instruction **"PostMessage(Wnd,WM_USER,0,0);"**qui est exécutée (rappelons qu'elle met le message dans la fifo de la fenêtre et que donc OnMessage l'intercepte, ce que l'on vérifie dans les images d'exécution ci-après :



Enfin la fiche de droite se ferme lors du click sur le bouton "Envoyer le message" lorsque le radio-bouton close est sélectionné (car c'est le message WM_USER+1 qui est posté dans la fifo) :



Il est conseillé d'exécuter cet exemple sur sa machine, puis de le modifier en testant d'autres niveaux d'interception, en bloquant le traitement par suppression du mot inherited afin de bien se familiariser avec l'utilisation pratique du mécanisme de répartition des messages avec Delphi.

Chapitre 8.3 Créer un événement associé à un message

Plan du chapitre:

1. Rappel sur la construction d'un événement

2. Exemple de création d'événements dans un TEdit

2.1 Evènement OnColorChange
2.2 Evènement OnResize

1. Rappel sur la construction d'un événement

Nous avons déjà fourni une notice méthodologique de construction d'un événement au chapitre 5.5, rappelons rapidement en le complétant ce qu'il faut savoir pour élaborer un nouvel événement :

Voici les étapes qui interviennent dans la définition d'un événement :

- Définition du type de gestionnaire
- Déclaration de l'événement
- Appel de l'événement

Pour construire un nouvel événement dans ClasseA :

- Il nous faut d'abord définir un type pour l'événement : EventTruc
- Il faut ensuite mettre dans ClasseA une propriété d'événement : **property** OnTruc : EventTruc
- Il faut créer un champ privé nommé FOnTruc de type EventTruc en lecture et écriture qui servira de champ de stockage de la propriété OnTruc.
- Si l'on pense à la réutilisation de la classe par héritage : Il est utile de créer une méthode centralisatrice **protected** virtuelle d'appel du pointeur de méthode FOnTruc, que les développeurs construisant une nouvelle classe héritant de ClasseA redéfiniront dans leur class fille.

Précédemment nous avons vu comment définir deux événements sur une structure de données de pile Lifo, nous rappelons ci-dessous une partie du code que nous allons modifier :

```
type
  DelegateLifo = procedure ( Sender: TObject ;  s :string ) of  object ;

 ClassLifo = class (TList)
 private
    FOnEmpiler : DelegateLifo ;
    FOnDepiler : DelegateLifo ;
 public
   function  Est_Vide  : boolean ;
   procedure  Empiler (elt : string ) ;
   procedure  Depiler ( var  elt : string ) ;
   property  OnEmpiler : DelegateLifo  read  FOnEmpiler  write  FOnEmpiler ;
   property  OnDepiler : DelegateLifo  read  FOnDepiler  write  FOnDepiler ;
 protected
   procedure  EvtEmpiler (elt : string ) ; virtual;
   procedure  EvtDepiler (elt : string ) ; virtual;
end;
```

Définition du type de gestionnaire

Déclaration de l'événement

Méthodes centralisatrices pour l'appel de l'événement

Les appels au pointeur de méthode (qui pointe vers le futur gestionnaire de l'événement) se situent à des endroits stratégiques choisis pour représenter la survenue de l'événement choisi et ont lieu à travers la méthode virtuelle protected :

```
Implementation
procedure  EvtEmpiler (s : string ) ;
begin
   if  assigned(FOnEmpiler)  then
      FOnEmpiler ( self , s )
end;

procedure  EvtDepiler (s : string ) ;
begin
   if  assigned(FDeEmpiler)  then
      FOnDepiler ( self , s )
end;

procedure  ClassLifo.Depiler( var  elt : string ) ;
 begin
  if  not  Est_Vide  then
  begin
    elt :=string (self.First) ;
    self.Delete(0) ;
    self.Pack ;
    self.Capacity := self.Count ;
    EvtDepiler (elt)
   end
  end;

 procedure  ClassLifo.Empiler(elt : string ) ;
 begin
   self.Insert (0 , PChar(elt)) ;
   EvtEmpiler (elt)
 end;
```

Appel de l'événement

Appel de l'événement

Ceci représente la construction générale de tout événement, ce qui peut différer d'une construction à l'autre c'est la façon et le lieu de l'appel de l'événement. Nous allons voir comment utiliser les messages système pour créer des événements en particulier dans les TControl.

2. Création de deux nouveaux événements dans un TEdit

Mise en oeuvre sur l'adjonction de 2 évènements nouveaux à un composant déjà existant.

Enoncé : construire dans une classe dérivant des TEdit

- un événement OnColorChange qui permet de notifier à l'utilisateur un changement dans la couleur d'un TEdit et de programmer éventuellement la gestion de cet événement.
- un événement OnResize qui permet de signaler que le TEdit vient de voir changer sa taille (l'une de ses deux propriétés width ou height).

2.1 Evénement OnColorChange

2.1.1 Déclenchement de l'événement
2.1.2 Définition du gestionnaire
2.1.3 Déclaration de l'événement
2.1.4 Propriété d'accès à l'événement
2.1.5 Appel de l'événement par procédure
2.1.6 Le gestionnaire vide de l'événement

2.1.1 Déclenchement de l'événement

Comme nous ne sommes pas maître du code source de base des TEdit, il ne nous est pas possible de travailler comme dans l'exemple précédent de la pile Lifo. Il nous faut essayer de trouver s'il existe un message système adéquat et alors l'intercepter pour l'utiliser à nos fins.

Après observation de la classe des TEdit, nous n'avons pas trouvé de message système informant le TEdit que sa couleur vient de changer : soit nous devons abandonner ,soit nous cherchons un message qui se rapporte à l'action qui a lieu lorsque la couleur vient de changer. Nous remarquons que dans l' éventualité du changement de couleur du fond d'un contrôle, le système redessine le contrôle.

Nous optons donc pour la redéfinition l'interception du message système WM_PAINT qui est envoyé très regulièrement à la fenêtre d'un contrôle afin que celui-ci se redessine. Nous avons vu au chapitre précédent comment déclarer la méthode d'interception du dit message, nous décidons de nommer cette méthode WMPaintChgColor :

```
procedure WMPaintChgColor (var Msg : TWMpaint); message WM_PAINT;
```

Le type TWMPaint est défini par Delphi pour coder après transtypage un message système WM_PAINT

```
TWMPaint = packed record
  Msg: Cardinal;
  DC: HDC;
  Unused: Longint;
  Result: Longint;
end;
```

2.1.2 Définition du gestionnaire

Nous construisons un gestionnaire d'événements spécifiques, car nous souhaitons disposer en paramètre de la nouvelle couleur.

```
TEventColor = procedure (Sender: TObject; couleur:Tcolor) of object;
```

2.1.3 Déclaration de l'événement

```
private    FColorChange : TEventColor;
```

2.1.4 Propriété d'accès à l'événement

```
published
   property OnColorChange : TEventColor read FColorChange write FColorChange;
```

2.1.5 Appel de l'événement par une procédure centralisatrice

```
protected
   procedure ColorChange(coul:TColor);virtual;
```

2.1.6 Le gestionnaire vide de l'événement OnColorChange doit être valide

```
if Assigned(FColorChange) then
  FColorChange(self,coul)
```

2.2 Evénement OnResize

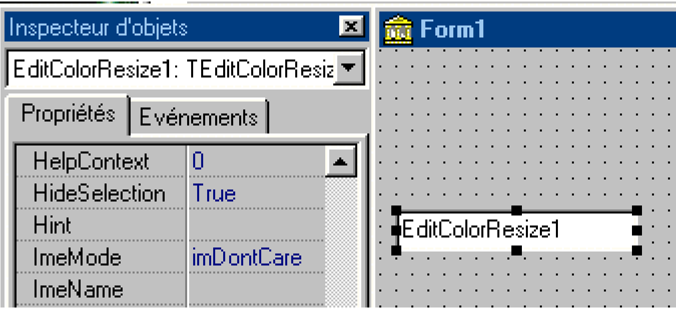
L'événement OnResize a été construit de la même manière que l'événement OnColorChange, par interception du message WM_SIZE, toutefois à titre d'exemple il n'y a pas de procédure surchargeable centralisatrice du genre **procedure** ColorChange. Dans ce cas le programmeur ne pourra pas rajouter un comportement spécifique sur l'événement OnResize. Nous ne répètons pas la démarche qui est strictement identique.

Ci-dessous, nous donnons le code source du composant simple dérivé des TEdit :

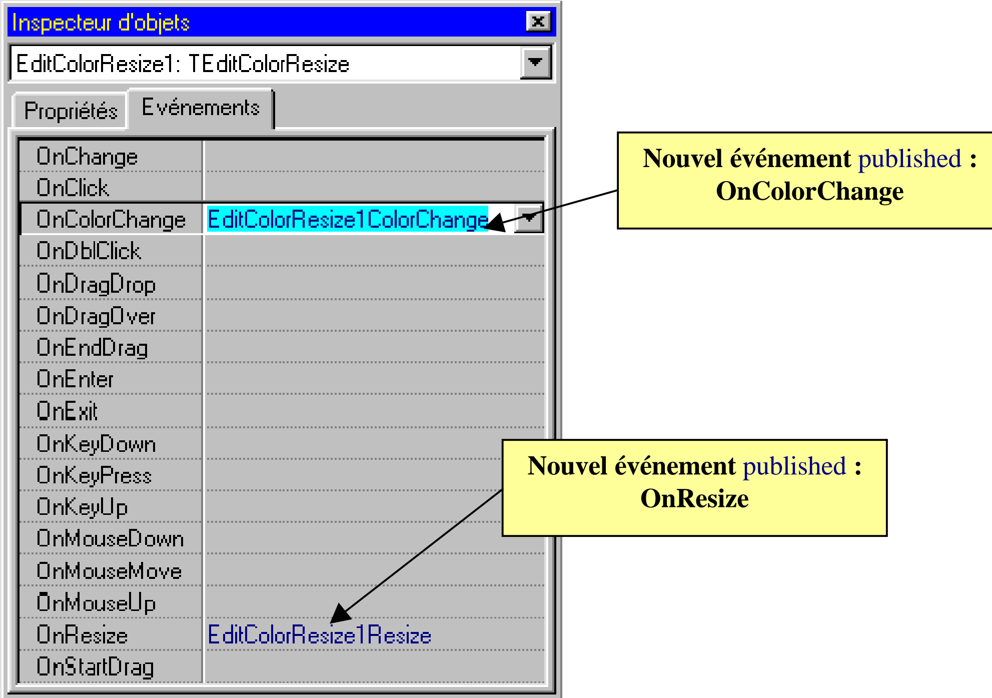
```
unit UEditColor;
interface

 uses
  SysUtils, Messages, Classes, Graphics, Controls,  Forms, StdCtrls, ExtCtrls, Outline;
type
  TEventColor = procedure (Sender: TObject; couleur:Tcolor) of object;
  { pointeur de gestionnaire d'événement OnColorChange (paramètre couleur : TColor, nécessaire ici }
  TEditColorResize = class(TEdit)
```

```
  private
   FColorChange : TEventColor;
  { pointeur de méthode spécifique permettant utiliser la property OnColorChange }
   ColorPrec:Tcolor; // la couleur précédente
  FOnResize:TNotifyEvent;
 { pointeur de méthode permettant utiliser la property OnResize }
   procedure WMPaintChgColor(var Msg:TWMpaint); message WM_PAINT;
 {le message WM_PAINT qui permettra de matérialiser l'événement}
  procedure WMSizeRedim(var Msg:TWMsize); message WM_size;
 {le message WM_size est envoyé au TEdit dès son height ou son width a changé }
 protected
    procedure ColorChange(coul:TColor);virtual;   {pour surcharge ultérieure par le programmeur }
 public
    constructor Create(AOwner: TComponent); override;
 published  //nouveau gestionnaire d'événement
    property OnColorChange : TEventColor read FColorChange write FColorChange;
    property OnResize:TNotifyEvent read FOnResize writeFOnResize;
 end;
procedure Register;

implementation

procedure Register;
begin
 RegisterComponents('Exemples', [TEditColorResize]);
end;
//-----------------------------------------------------------//
constructor TEditColorResize.Create(AOwner: TComponent);
begin
 inherited;
 color:=clwhite;
 ColorPrec:=self.color // initialisation à la couleur actuelle
end;
procedure TEditColorResize.WMSizeRedim(var Msg:TWMsize);
// intercepte le message WM_size et traite le changement de taille
begin
 if Assigned(OnResize) then
   FOnResize(self) //lien avec le futur gestionnaire du programmeur
end;
procedure TEditColorResize.WMPaintChgColor(var Msg:TWMpaint);
// intercepte le message WM_Paint et traite le changement de couleur
begin
 inherited ;
 if color<>ColorPrec then
  begin
   ColorChange(ColorPrec);
   ColorPrec:=color;
   change // appelle le gestionnaire d'événement OnChange, s'il est défini.
  end
end;
procedure TEditColorResize.ColorChange(coul:TColor);
// pour que le développeur puisse la surcharger éventuellement
begin
  if Assigned(FColorChange) then
   FColorChange(self,coul) // lien avec le futur gestionnaire du programmeur
end;
end.
```

Enregistrez ce composant dans votre Delphi, puis testez-le sur les deux événements nouveaux en écrivant un projet de test et en programmant les deux gestionnaires d'événements.
Voici après dépôt d'un TEditColorResize ce que l'inspecteur d'objet nous montre sur cette classe:



toutes les propriétés sont celles des TEdit



Les 2 nouveaux événements (OnColorChange et OnResize)

Nous montrons ci-dessous comment un développeur utilisant notre composant TEditColorResize peut surcharger et implanter un comportement additionnel à l'événement *OnColorChange.*

```
interface
TEditUser = class(TEditColorResize)
  protected
   procedure ColorChange(coul:TColor);override;
end;
implementation
procedure TEditUser.ColorChange(coul:TColor);
begin
 inherited; // appel à la procédure centralisée de la classe ancêtre
 //...comportement nouveau rajouté par le programmeur
end;
```

Chapitre 8.4 ActiveX avec la technologie COM

Plan du chapitre:

Introduction

1. Les notions d'interfaces COM de microsoft

La notion d'interface en général avec Delphi
1.1 Qu'est-ce qu'une interface COM
1.2 L'interface Iunknown
1.3 Les CoClasses en Delphi
1.4 l'architecture COM et le registre Windows

2. ActiveX est un objet COM

3. Création d'un ActiveX avec Delphi

1°) Construction à partir du composant
2°) Afficher l'expert contrôle ActiveX
3°) Recenser l'ActiveX
4°) Comment installer dans la palette Delphi un ActiveX

4.Déploiement et utilisation Web d'une fiche ActiveX

1°) Construction de la fiche de base
2°) Recensement de la fiche ActiveX
3°) Déploiement sur le web

Introduction

COM signifie **C**omponent **O**bject **M**odel, c'est le modèle proposé par Microsoft pour créer des composants logiciels distribués. La principale fonction de COM que nous voyons dans ce chapitre consiste à construire des composants fenêtrés nommés ActiveX qui peuvent être utilisés sur Internet à travers "Internet Explorer" et développés et manipulés par des langages différents. Les ActiveX peuvent être développés comme les composants Delphi, dans un environnement de développement Delphi, C++, VB6,… et être réutilisés dans l'un quelconque de ces environnements. Les ActiveX peuvent aussi être convertis en Winforms dans la plate-forme .**Net** et être ainsi réutilisés par les langages de .**Net** comme : C#, VB Net,…

La famille de systèmes d'exploitation Windows utilise cette architecture, et de très nombreux développements ont été centrés sur cette norme. L'architecture .**Net** de microsoft se positionne comme remplaçant de COM tout en gardant dores et déjà une compatibilité ascendante avec celle-ci.

1. Les notions d'interfaces COM de microsoft

La notion d'interface en général avec Delphi

- Une interface est un contrat, elle peut contenir des **propriétés**, des **méthodes** mais **ne** doit contenir **aucun champ** ou **attribut**.
- Une interface **ne** peut **pas** contenir des méthodes déjà implémentées.
- Tous les membres d'une interface sont **publics** sans nécessiter de qualificateur de visibilité.
- Une interface est héritable.
- Pour pouvoir construire un objet à partir d'une interface, il faut définir une classe non abstraite implémentant **toutes** les méthodes de l'interface.
- En Delphi une interface se déclare avec le mot clef **interface**, aux mêmes endroits qu'une déclaration de classe.

Exemple de déclaration d'interface en Delphi :

Animal = **Interface**
    procedure SeDeplacer;
    procedure Manger;
    function NombreDePattes : integer;
end;
	Indique des méthodes de fonctionnement d'un animal :
	❑ Comment se déplace-t-il
	❑ Comment mange-t-il
	❑ Combien a-t-il de pattes,

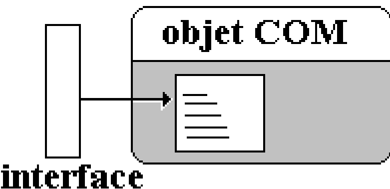
Oiseau = **Interface** (Animal) **procedure** ConstruireNid; **end;**	Un oiseau possédera les 3 méthodes précédentes + une méthode expliquant comment il construit son nid.

1.1 Qu'est-ce qu'une interface COM

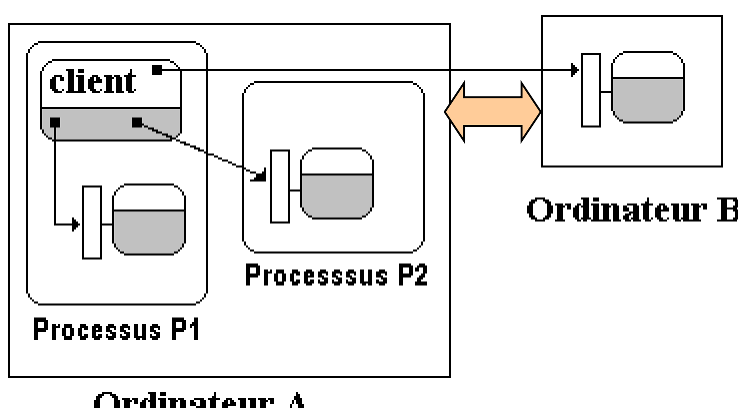
Les clients COM communiquent avec des objets par le biais d'interfaces COM. Les interfaces sont des groupes de routines, qui assurent la communication entre le fournisseur d'un service (objet serveur) et ses objets clients.

Rappelons qu'une interface représente juste un *contrat*, c'est-à-dire que la classe chargée d'implémenter cette interface s'engage à implémenter **toutes** les méthodes déclarées dans l'interface. **L'interface est l'unique moyen** de mettre à disposition du client le service fourni par l'objet, sans lui donner les détails de l'implémentation sur la façon dont ce service est fourni.

Une interface COM est **un pointeur dans un objet COM**, elle pointe en fait sur la table des méthodes de l'objet COM. Une interface n'est donc pas un objet, elle est en outre **identifiée de manière unique** par un nombre appelé GUID.



Avec COM, un client n'a pas besoin de savoir où réside l'objet serveur qu'il invoque (que l'objet réside dans le même processus P1 que le client, dans un autre processus P2 sur la machine A du client ou sur une autre machine B du réseau). C'est COM qui met en place les moyens d'accéder à l'objet serveur.



1.2 L'interface IUnknown

Nous avons précisé plus haut qu'afin d'obtenir des objets indépendants (du langage, de la plate-forme d'exécution,...) nous devions pouvoir disposer d'une interface de base commune ou universelle.

L'interface **IUnknown** est l'interface de base commune et universelle du **modèle COM.** Elle doit être présente dans tout langage implantant l'architecture COM.

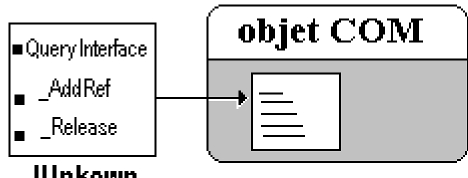
En Delphi toute interface hérite directement ou indirectement de **Iunknown.** (c'est d'ailleurs ce qui rend la notion d'interface plus lourde qu'en Java ou C#)

En Delphi voici la déclaration de **IUnknown** qui possède seulement 3 méthodes :

```
type
IInterface = interface
['{00000000-0000-0000-C000-000000000046}']
  function QueryInterface (const IID: TGUID; out Obj): HResult; stdcall;
  function _AddRef: Integer; stdcall;
  function _Release: Integer; stdcall;
end;

IUnknown = IInterface;
```

QueryInterface	permet de naviguer entre les différentes interfaces implémentées par un objet COM (comme toute interface dispose de cette méthode, on peut accéder à n'importe qu'elle interface à partir de IUnknown).
_AddRef	permet l'incrémentation d'un compteur de références : : lorsqu'un objet se connecte à l'interface, le compteur est incrémenté de 1.
_Release	permet la décrémentation du compteur de référence : lorsqu'un objet se déconnecte de l'interface, le compteur est décrémenté de 1. Ainsi, lorsque le compteur est arrivé à 0, l'objet COM sait qu'il peut libérer la mémoire qu'il occupait.

Chaque langage de programmation (C++, Delphi,...) se charge ensuite d'implanter l'interface **IUnknown** afin d'instancier un objet COM qui dérive obligatoirement et au minimum cette interface **IUnknown**.



En Delphi, c'est la classe **TInterfacedObject** qui est chargée d'implémenter l'interface **IUnknown**. Elle est déclarée comme suit :

```
TInterfacedObject = class(TObject, IInterface)
 protected
   FRefCount: Integer;
   function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
   function _AddRef: Integer; stdcall;
   function _Release: Integer; stdcall;
 public
   procedure AfterConstruction; override;
   procedure BeforeDestruction; override;
   class function NewInstance: TObject; override;
   property RefCount: Integer read FRefCount;
end;
```

1.3 Les CoClasses en Delphi

Dans Delphi un objet COM est nécessairement comme tout objet de Delphi, une instanciation d'une classe Delphi. Pour un objet COM cette classe doit obligatoirement être une classe très spéciale : une **coclasse**.

Un objet COM est en Delphi une instance d'une CoClasse, qui est une classe implémentant une ou plusieurs interfaces COM. L'objet COM fournit les services définis par les méthodes de ses interfaces.

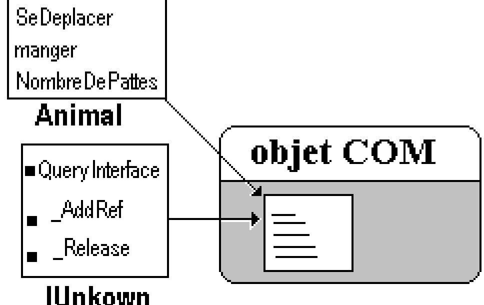
> Une coclasse est une classe chargée d'implémenter une ou plusieurs interfaces. C'est elle qui va "contenir" le code machine et donc constituer le moule de fabrication de l'objet COM.
>
> Une coclasse peut bien sûr dériver d'une classe existante.
>
> Une coclasse doit dériver d'une classe implantant l'interface **IUnknown**.

Voici un exemple de déclaration d'une coclasse :

```
Animal = Interface // hérite automatiquement de IUnknown
    procedure SeDeplacer ;
    procedure Manger;
    function NombreDePattes : integer;
end;

oiseau = class ( Tobject, Animal )
  public
     procedure SeDeplacer ;
     procedure Manger;
     function NombreDePattes : integer;
   private
     function DureeCouvaison : integer ;
end;
```

Il faut implémenter les 6 méthodes : SeDeplacer, Manger, NombreDePattes, QueryInterface, _AddRef , _Release + la nouvelle méthode DureeCouvaison.

Ci-dessous un objet COM de type oiseau :



Toute coclasse dérivant de **TInterFacedObject** hérite de l'implémentation de **IUnknown**. Il est donc conseillé de construire les coclasses permettant d'instancier des objets COM à partir de cette classe.

```
 Animal = Interface // hérite automatiquement de IUnknown
     procedure SeDeplacer ;
     procedure Manger;
     function NombreDePattes : integer;
end;

oiseau = class (TInterFacedObject, Animal )
  public
       procedure SeDeplacer ;
       procedure Manger;
       function NombreDePattes : integer;
   private
      function DureeCouvaison : integer ;
end;
```

Il suffit alors d'implémenter les 3 méthodes : SeDeplacer, Manger, NombreDePattes + la nouvelle méthode DureeCouvaison.

1.4 l'architecture COM et le registre Windows

Pour assurer l'unicité des identificateurs d'objets COM, on utilise des GUIDs (**Globally Unique IDentifiers**), qui comme leur nom l'indique permettent d'avoir des identifiants (presque) uniques au monde.

Ces GUIDs sont codés sur 128 bits et sont souvent représentés sous une forme standard en hexadécimal afin de les rendre plus "lisibles" à l'homme :

Exemple de GUID

{ 27E14E2B-1104-44B2-AE49-9A8778A130EA}

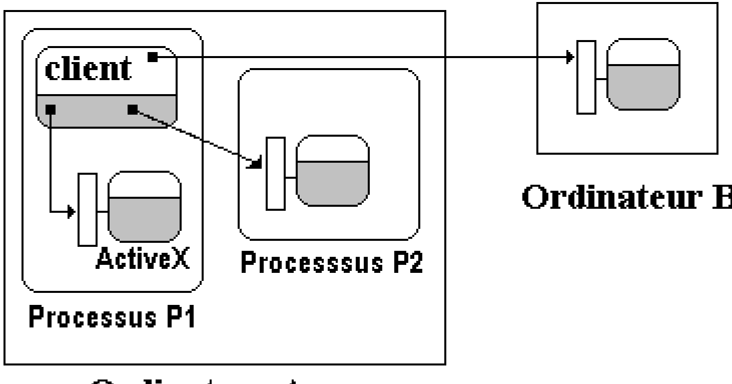
Un GUID peut servir à identifier un objet COM (on parle alors de **CLSID** ou **cl**ass **ID**), une interface (IID), ou bien tout élément que l'on doit identifier très précisément (par exemple une instance d'application).Tout objet COM, pour pouvoir être instancié, doit être au préalable inscrit dans le registre système de Windows (grâce à un utilitaire comme regsvr32.exe du système windows ou tregsvr, qui est livré avec Delphi).

2.ActiveX est un objet COM

Un contrôle ActiveX est un composant logiciel COM qui est inclus dans une application hôte capable de gérer les contrôles ActiveX. Un contrôle ActiveX "étend" les fonctionnalités de l'application conteneur. un tel composant ne nécessite que l'interface **IUnknown**.

Les contrôles **ActiveX** sont conçus pour être eux-mêmes incorporés dans une application client, **ils s'exécutent alors dans le même espace processus** que le client.

Ci-dessous l'application client fait partie du processus P1 ainsi que l'ActiveX qu'elle utilise (les autres objets COM auxquels elle accède ne sont pas des ActiveX). C'est exactement ce qui se passe lorsque l'on "exécute" un ActiveX dans le navigateur "Internet Explorer" :

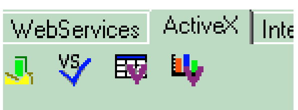


Un contrôle **ActiveX** en Delphi doit au minimum implanter certaines interfaces nécessaires, dont voici la liste fournit par Borland :

IUnknown, IDispatch, IPersistStreamInit, IOleInPlaceActiveObject, IPersistStorage, IViewObject, IOleObject, IViewObject2, IOleControl, IPerPropertyBrowsing, IOleInPlaceObject, ISpecifyPropertyPages .

Une fois conçu, il pourra être utilisé comme n'importe quel autre composant classique de la VCL de Delphi pour être incorporé à l'application.

La palette des composants de Delphi fournit en standard quatre contrôles ActiveX (certains ont été écrits en C++, d'autre en VB6):



Les paragraphes suivant traitent plus spécialement de méthodes pratiques pour construire des ActiveX en Delphi et le déploiement sur le web pour l'utilisation internet d'unActiveX.

3 Création d'un ActiveX avec Delphi

Delphi fournit depuis la version 4, **heureusement** un **expert de construction des ActiveX** qui implantent incluent et génèrent **automatiquement** le code source associé aux diverses interfaces nécessaires. Le seul travail restant à effectuer est de programmer les nouvelles fonctionnalités du futur ActiveX.

Un contrôle ActiveX dans Delphi est simplement un contrôle VCL encapsulé dans une enveloppe de classe ActiveX.

La démarche à suivre est très simple, l'expert effectuant automatiquement toutes les opérations il suffit de suivre et de répondre aux questions de celui-ci, le seul vrai travail consiste à développer un composant Delphi!

Utiliser l'expert contrôle ActiveX

Pour créer un nouveau contrôle ActiveX à partir d'un contrôle VCL personnalisé, Delphi propose deux experts permettant de construire :

> *Des contrôles ActiveX qui encapsulent des classes VCL.*
> *Des fiches ActiveX semblables aux contrôles, mais dirigées vers le déploiement Web.*

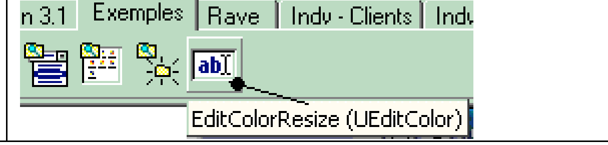
Dans les deux cas l'expert place en fait une enveloppe de classe ActiveX autour d'un contrôle VCL ou d'une fiche et construit le contrôle ActiveX qui contient l'objet.

Construisons pas à pas , en 4 étapes, un ActiveX

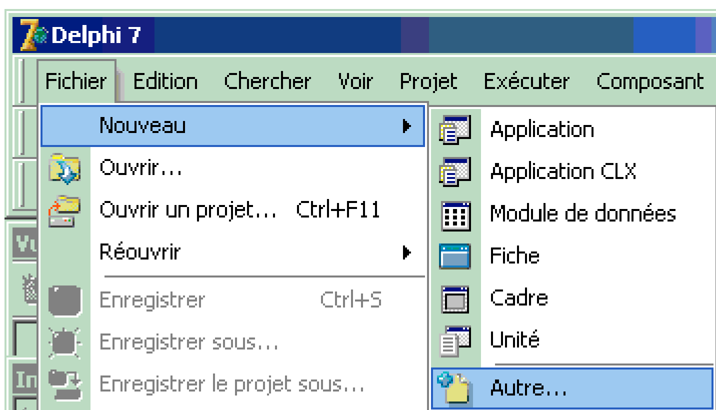
1°) Construction à partir du composant TEditColorResize du sous-chapitre précédent :

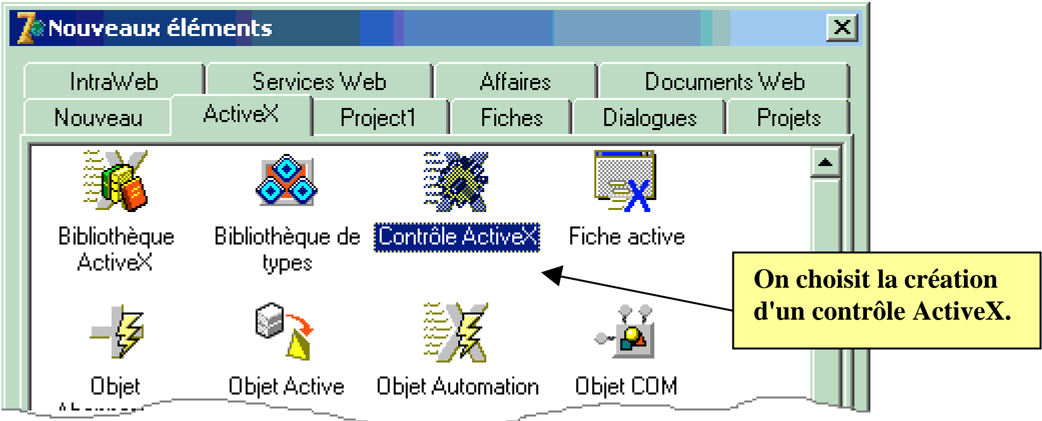
```
unit UEditColor;
interface

 uses
  SysUtils, Messages, Classes, Graphics, Controls,  Forms, StdCtrls, ExtCtrls, Outline;
type
  TEventColor = procedure (Sender: TObject; couleur:Tcolor) of object;
  { pointeur de gestionnaire d'événement OnColorChange (paramètre couleur : TColor, nécessaire ici }
  TEditColorResize = class(TEdit)
   private
    FColorChange : TEventColor;
   { pointeur de méthode spécifique permettant d'utiliser la property OnColorChange }
    ColorPrec:Tcolor; // la couleur précédente
   FOnResize:TNotifyEvent;
  { pointeur de méthode permettant utiliser la property OnResize }
    procedure WMPaintChgColor(var Msg:TWMpaint); message WM_PAINT;
  {le message WM_PAINT qui permettra de matérialiser l'événement}
   procedure WMSizeRedim(var Msg:TWMsize); message WM_size;
  {le message WM_size est envoyé au TEdit dès son height ou son width a changé }
  protected
     procedure ColorChange(coul:TColor);virtual;   {pour surcharge ultérieure par le programmeur }
  public
     constructor Create(AOwner: TComponent); override;
  published  //nouveau gestionnaire d'événement
     property OnColorChange : TEventColor read FColorChange write FColorChange;
     property OnResize:TNotifyEvent read FOnResize writeFOnResize;
  end;
procedure Register;

implementation
```

```
  procedure Register;
  begin
   RegisterComponents('Exemples', [TEditColorResize]);
  end;
  //-----------------------------------------------------------//
  constructor TEditColorResize.Create(AOwner: TComponent);
  begin
   inherited;
   color:=clwhite;
   ColorPrec:=self.color // initialisation à la couleur actuelle
  end;
  procedure TEditColorResize.WMSizeRedim(var Msg:TWMsize);
  // intercepte le message WM_size et traite le changement de taille
  begin
   if Assigned(OnResize) then
     FOnResize(self) //lien avec le futur gestionnaire du programmeur
  end;
  procedure TEditColorResize.WMPaintChgColor(var Msg:TWMpaint);
  // intercepte le message WM_Paint et traite le changement de couleur
  begin
   inherited ;
   if color<ColorPrec then
    begin
     ColorChange(ColorPrec);
     ColorPrec:=color;
     change // appelle le gestionnaire d'événement OnChange, s'il est défini.
    end
  end;
  procedure TEditColorResize.ColorChange(coul:TColor);
  // pour que le développeur puisse la surcharger éventuellement
  begin
    if Assigned(FColorChange) then
     FColorChange(self,coul) // lien avec le futur gestionnaire du programmeur
  end;
end.
```

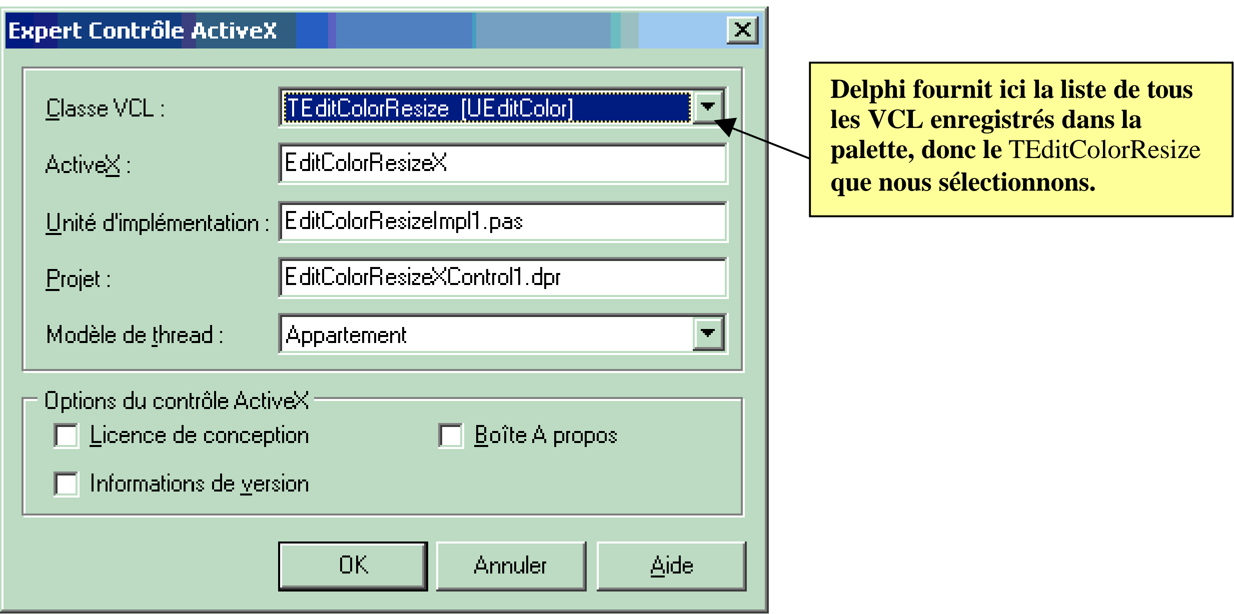
Composant que nous avons rangé dans la palette des composants dans l'onglet exemples :	

2°) Afficher l'expert contrôle ActiveX (à partir du menu Fichier\Nouveau\Autre) :





Delphi envoie un formulaire de saisie d'information sur le paramétrage de création du contrôle ActiveX, nous lui indiquons que nous voulons encapsuler le composant TEditColorResize :



L'expert génère automatiquement des fichiers intermédiaires dont :

Une bibliothèque ActiveX contenant le code nécessaire au démarrage d'un contrôle ActiveX (21 lignes)

```
library EditColorResizeXControl1;

 uses
  ComServ,
  EditColorResizeXControl1_TLB in 'EditColorResizeXControl1_TLB.pas',
  EditColorResizeImpl1 in 'EditColorResizeImpl1.pas' {EditColorResizeX: CoClass};
   {$E ocx}
  exports
              DllGetClassObject,
              DllCanUnloadNow,
              DllRegisterServer,
              DllUnregisterServer;
  {$R *.TLB}
  {$R *.RES}

  begin
  end.
```

Une unit d'ActiveX descendant de TActiveXControl encapsulant le TEditColorResize (560 lignes)

```
unit EditColorResizeImpl1;

 {$WARN SYMBOL_PLATFORM OFF}

 interface

 uses
  Windows, ActiveX, Classes, Controls, Graphics, Menus, Forms, StdCtrls,
  ComServ, StdVCL, AXCtrls, EditColorResizeXControl1_TLB, UEditColor;

 type
  TEditColorResizeX = class(TActiveXControl, IEditColorResizeX)
 private
 { Déclarations privées }
 FDelphiControl: TEditColorResize;
 FEvents: IEditColorResizeXEvents;
  procedure ChangeEvent(Sender: TObject);
  procedure ClickEvent(Sender: TObject);
  procedure ColorChangeEvent(Sender: TObject; couleur: TColor);
  procedure DblClickEvent(Sender: TObject);
  procedure KeyPressEvent(Sender: TObject; var Key: Char);
  procedure ResizeEvent(Sender: TObject);
 protected
 { Déclarations protégées }
  procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage); override;
  procedure EventSinkChanged(const EventSink: IUnknown); override;
  procedure InitializeControl; override;
  function DrawTextBiDiModeFlagsReadingOnly: Integer; safecall;
  function Get_AlignDisabled: WordBool; safecall;
  function Get_AutoSelect: WordBool; safecall;
  function Get_AutoSize: WordBool; safecall;
  function Get_BevelInner: TxBevelCut; safecall;
  function Get_BevelKind: TxBevelKind; safecall;
  .........
end;

implementation
 .........
end.
```

L'expert génère aussi : une bibliothèque de type (EditColorResizeXControl1_TLB.pas) qui définit une CoClasse pour le contrôle (540 lignes).

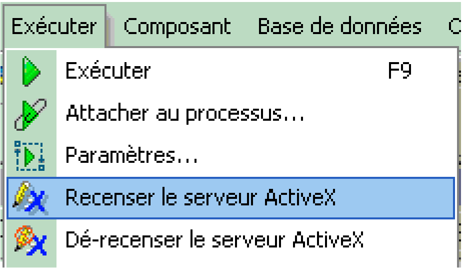
Si nous décomptons les lignes que l'expert a engendré pour encapsuler notre composant, nous obtenons un total de 1120 lignes ! Cela montre malgré tout la lourdeur du codage COM, heureusement nous n'avons pas à intervenir sur ces lignes de codes. Il nous reste à rendre accessible notre nouvel ActiveX afin que d'autres applications puissent l'utiliser.

3°) Recenser l'ActiveX nouvellement créé

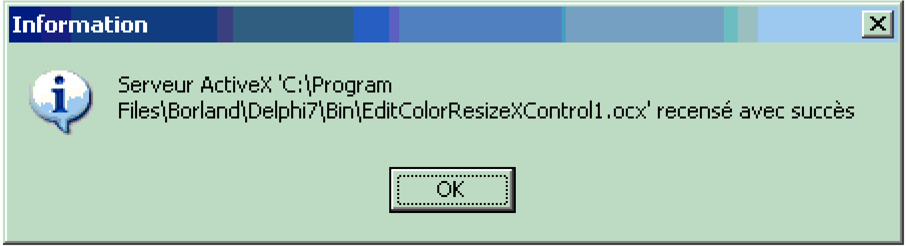
Une fois le contrôle ActiveX créé, il faut le recenser afin que d'autres applications puissent le trouver et l'utiliser c'est à dire l'inscrire dans la base des registres de windows comme serveur

COM. On dispose soit du **regsvr32.exe …** du système d'exploitation (en fenêtre de commande DOS), soit de la commande recenser dans le menuExécuter de Delphi :

Menu Exécuter/Recenser le serveur ActiveX :



Delphi répond :

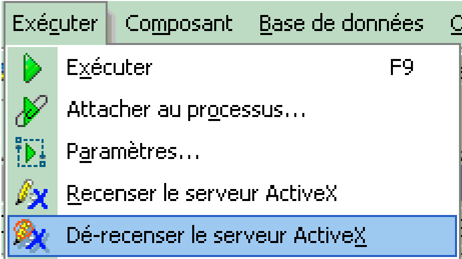


Au final Delphi engendre un seul fichier utile : le fichier contenant l'ActiveX. Les fichiers ActiveX se terminent par le suffixe OCX.

Dans le cas de notre exemple le fichier engendré a pour nom : **EditColorResizeXControl1.ocx.**

Vous pouvez mettre ce fichier où vous le voulez, pour pouvoir l'utiliser il faut le recenser là où vous l'avez mis

Remarque :
Avant de supprimer un contrôle ActiveX du système, il faut annuler son recensement avec le menu *Exécuter/Dérecenser le serveur ActiveX*.



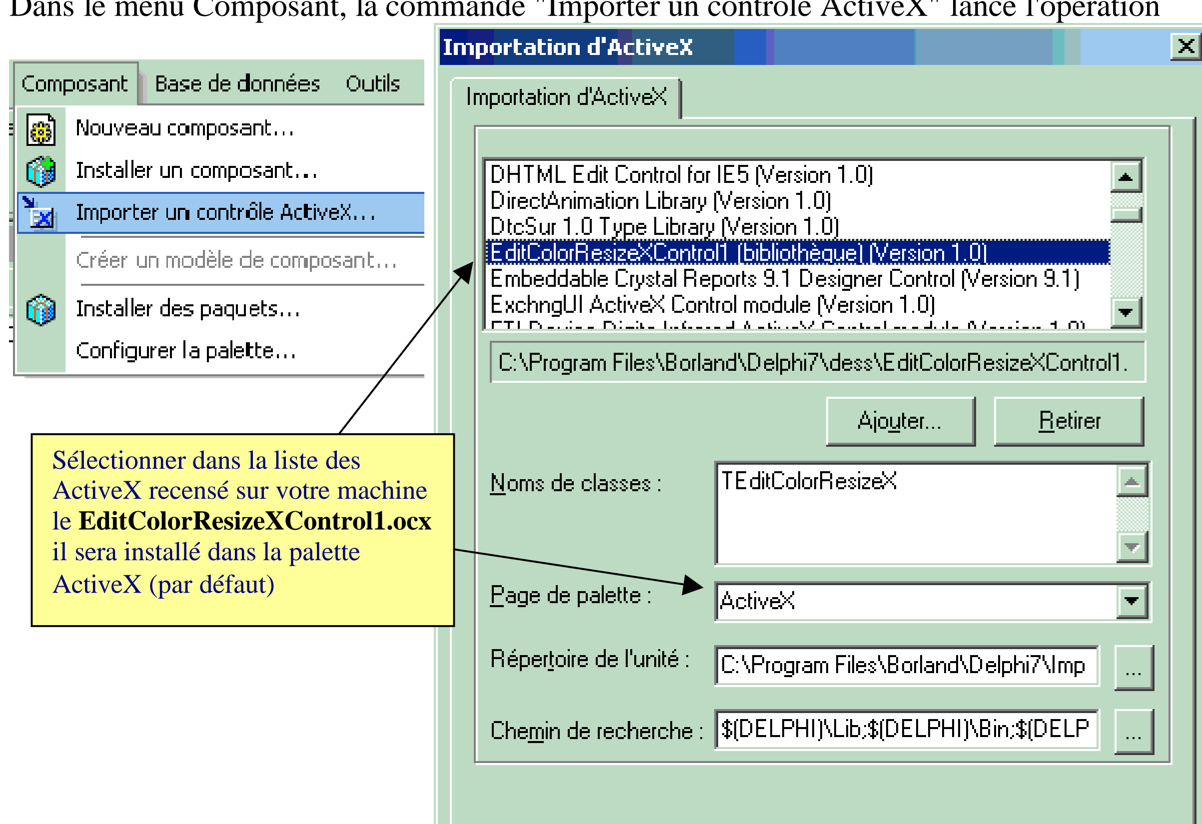
Ou bien lancer la commande **regsvr32.exe /u …** du système d'exploitation (en fenêtre de commande DOS)

Le composant **EditColorResizeXControl1.ocx** est maintenant utilisable avec un autre environnement C++, visual Basic,…

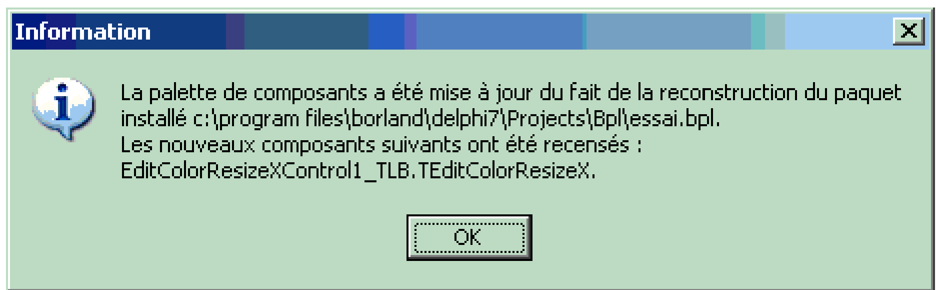
Afin de vérifier le bon fonctionnement de **EditColorResizeXControl1.ocx** nous allons l'installer dans la palette des composants Delphi comme un nouvel ActiveX (Delphi ignore qu'il a été conçu par Delphi, c'est un ActiveX comme des milliers d'autres que vous pouvez acheter ou récupérer gratuitement sur Internet).

4°) Comment installer dans la palette Delphi un ActiveX

Dans le menu Composant, la commande "Importer un contrôle ActiveX" lance l'opération



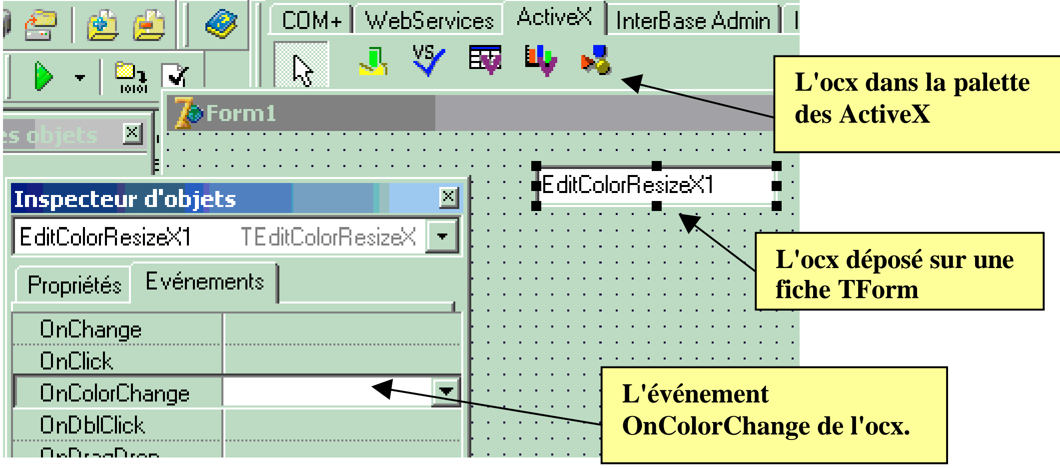
La suite du dialogue avec Delphi est identique à celle que vous avez lorsqu'il s'agit d'installer un composant Delphi classique, le dialogue se termine par l'affichage de la boîte suivante :



A partir de cet instant lorsque vous avez validé l'installation dans la palette vous disposez de la même entité représentée sous deux formats différents :

Un éditeur mono-ligne sensible au changement de couleur et au redimensionnement, vous en avez un version VCL (donc utilisable uniquement avec Delphi) et une autre version ocx utilisable avec tous les environnements de développements qui acceptent les ActiveX.

Enfin ce composant est converti par Net Framework à travers l'outil de conversion :
aximp c:\....\ EditColorResizeXControl1.ocx → en un fichier **AxEditColorResizeXControl1.dll** utilisable par tout langage de l'architecture Net.



4. Déploiement et utilisation Web d'une fiche ActiveX

Delphi propose un expert pour le déploiement web d'une fiche ActiveX :

> L'expert ActiveForm permet de créer de toutes pièces une nouvelle application ActiveX. L'expert configure le projet et ajoute une fiche vide dont on peut commencer la conception en ajoutant des contrôles.

La coclasse TActiveFormX = class(TActiveForm, IActiveFormX) représente une fiche ActiveX, c'est un contrôle ActiveX fondé sur une fiche VCL sur laquelle vous pouvez déposer n'importe lequel des composants de la palette de Delphi d'où un gain de temps de développement.

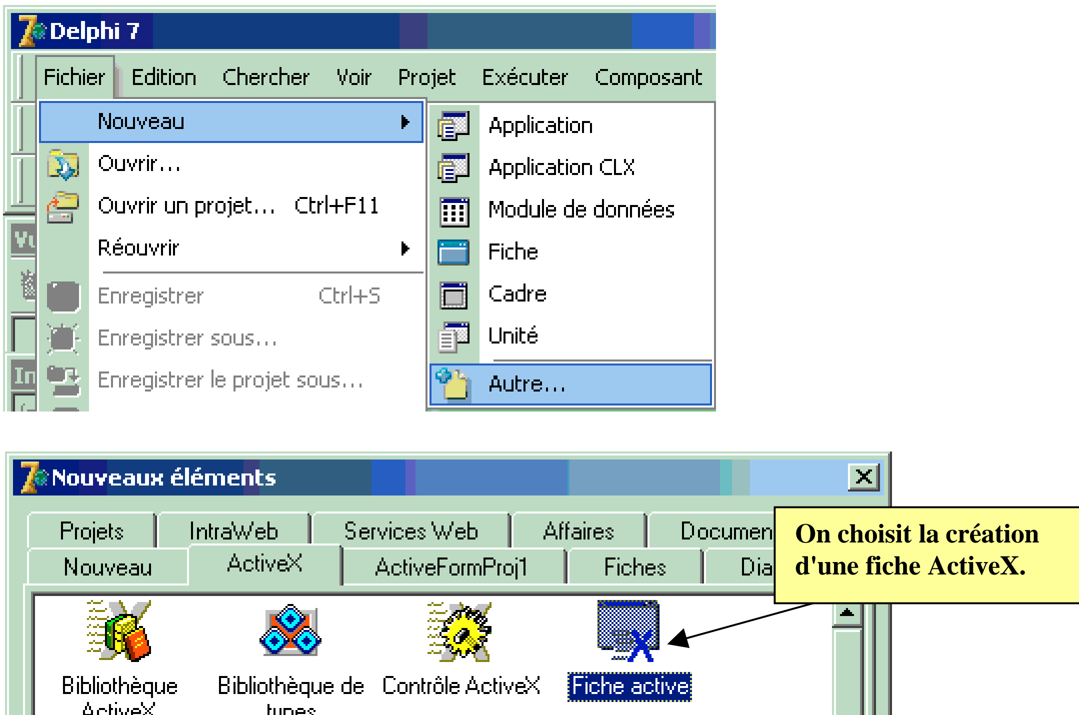
La fiche ActiveX ainsi créée peut être déployée sur le Web : la fiche ActiveForm est ensuite affichable et exécutable à l'intérieur d'une page html dans un navigateur Web (Internet Explorer). Dans le navigateur Web, la fiche se comporte comme une application autonome et peut mettre en œuvre des actions complexes. Les fiches ActiveX sont les "concurrents" (mono-plateforme) des applets Java (multi-plateforme).

Utiliser l'expert ActiveForm pour créer une Fiche ActiveX basée sur une fiche VCL

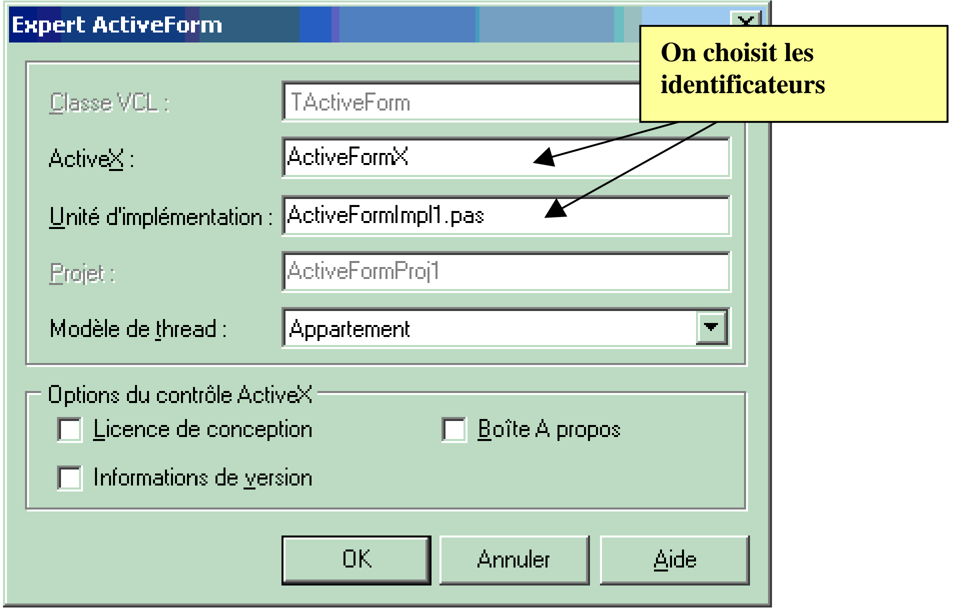
La démarche est sensiblement la même que pour créer un contrôle ActiveX, nous allons procéder à une création d'une telle fiche pas à pas sur un exemple.

Construisons pas à pas , en 4 étapes, une fiche ActiveX

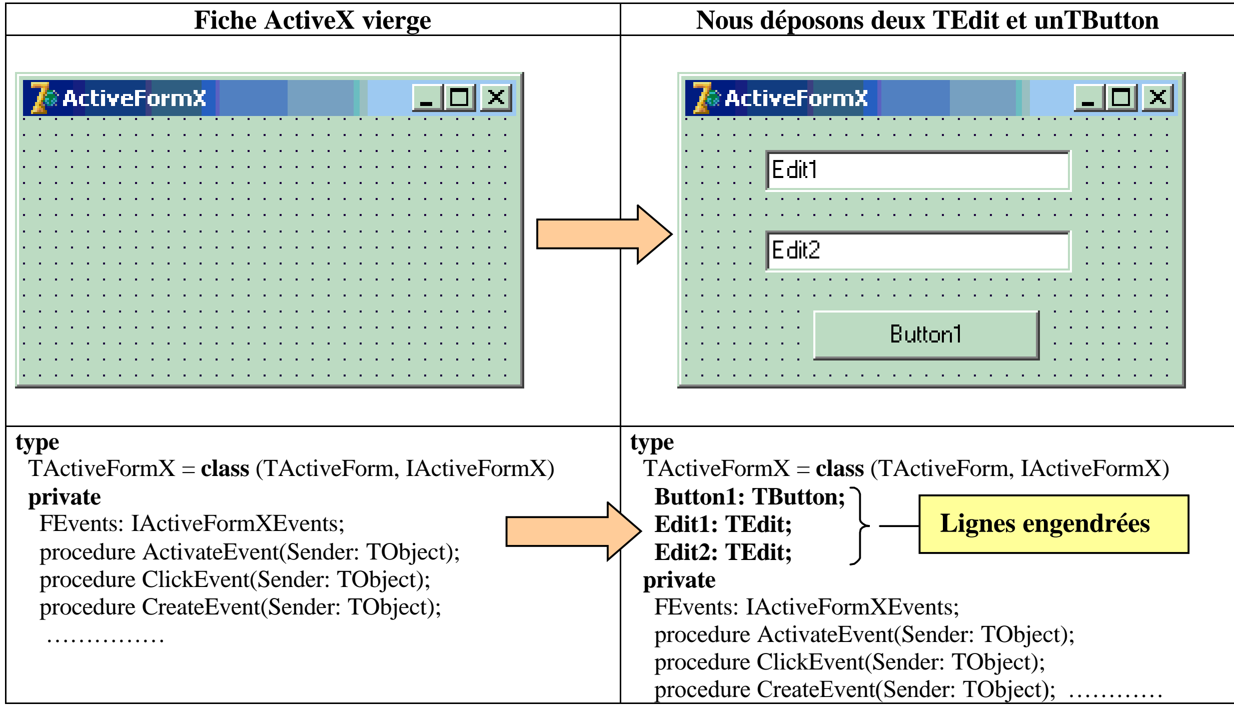
1°) Construction de la fiche de base



Delphi envoie un formulaire de saisie d'information sur le paramétrage de création du contrôle ActiveX qui hérite obligatoirement de TActiveForm, nous lui indiquons éventuellement le nom de la classe et celui de la unit (nous laissons le modèle de thread par défaut) :



A ce stade, l'expert génère comme au paragraphe précédent, 3 fichiers intermédiaires (de 21 +349 +334 = 704 lignes) et propose une fiche visuelle vierge sur laquelle on peut insérer des contrôles et travailler avec la fiche comme on le ferait pour toute application Delphi.

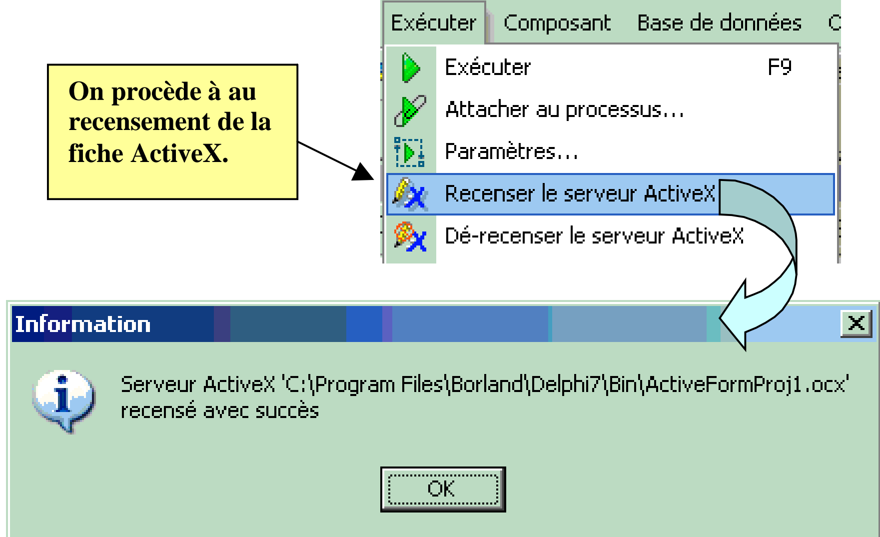


Nous programmons la recopie du texte de l'Edit2 dans l'Edit1 lorsque l'on clique sur le Button1, comme pour une application, Delphi nous propose le squelette du gestionnaire d'événement OnClick du Button1 que nous remplissons avec notre ligne de code de copie :

```
procedure TActiveFormX.Button1Click(Sender: TObject);
begin
  Edit1.Text:=Edit2.Text
end;
```

Nous supposons nous arrêter à ce stade et passer à la suite de la création de l'ActiveX.

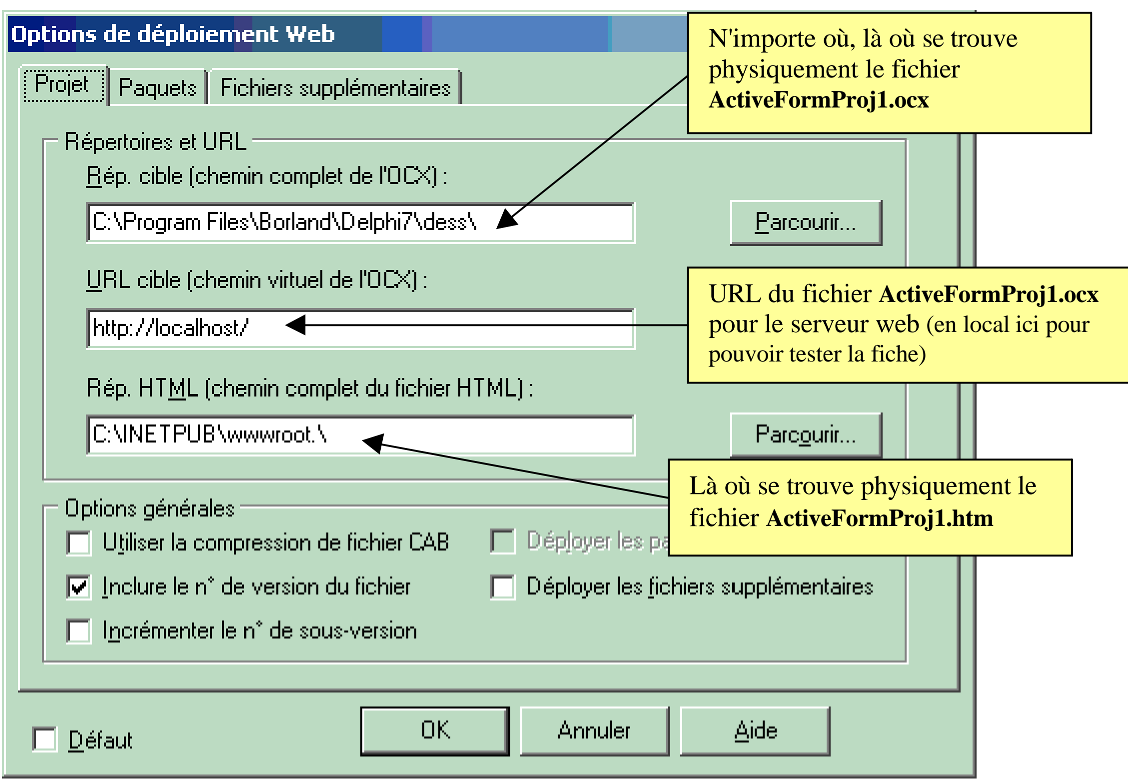
2°) Recensement de la fiche ActiveX ainsi construite (ActiveFormProj1.ocx)



3°) Déploiement sur le web

Avant de pouvoir utiliser dans un client Web, comme Internet Explorer, la fiche ActiveFormProj1.ocx ainsi créée, il faut la déployer sur le serveur. A chaque fois que la fiche ActiveX est modifiée, elle doit être recensée de nouveau et re-déployée afin que les modifications soient effectives.

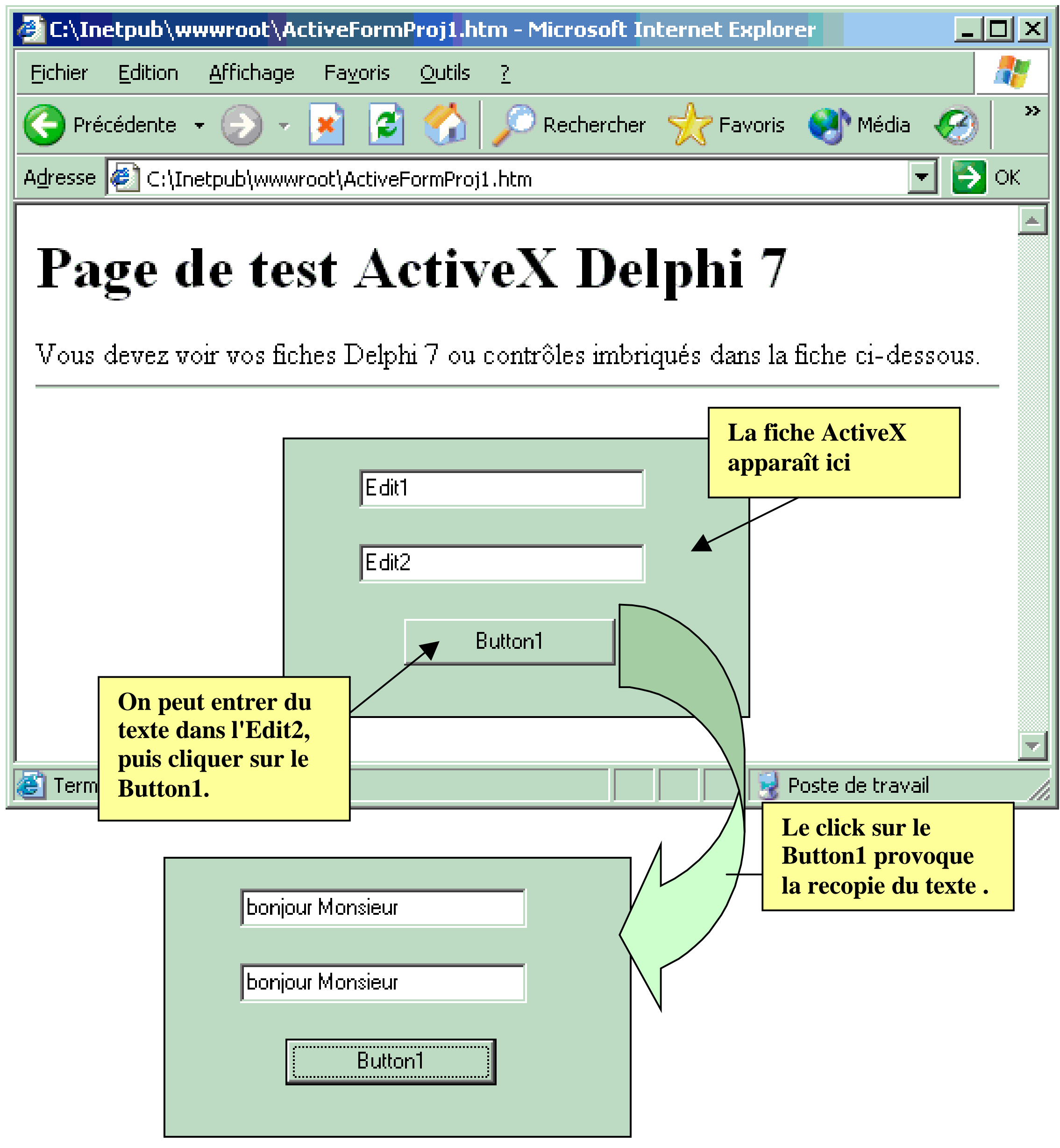
Pour déployer un contrôle ActiveX, il est nécessaire de disposer d'un serveur Web Web (IIS pour notre exemple). Par contre, il n'est pas obligatoire d'enregistrer le contrôle dans la palette de composant puisque, en général, le composant web créé est dédié à une application web particulière.



Delphi engendre un fichier **ActiveFormProj1.htm** permettant de tester la fiche ActiveX :

```
<HTML>
<H1> Page de test ActiveX Delphi 7 </H1><p>
Vous devez voir vos fiches Delphi 7 ou contrôles imbriqués dans la fiche ci-dessous.
<HR><center><P>
<OBJECT
        classid="clsid:F9A089C7-B331-404E-B21F-4541846FA390"
        codebase="http://localhost/ActiveFormProj1.ocx#version=1,0,0,0"
        width=350
        height=250
        align=center
        hspace=0
        vspace=0 >
</OBJECT>
</HTML>
```

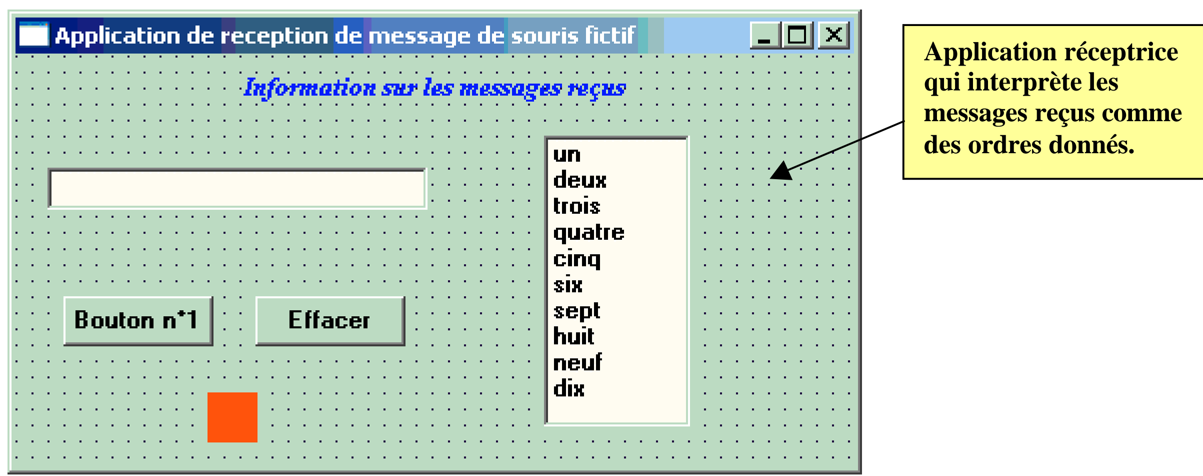
Si vous avez installé un serveur web sur votre machine IIS par exemple alors l'exécution du fichier **ActiveFormProj1.htm** dans le navigateur IE donne la page html de test suivante :



Quand cette page HTML est visualisée dans le navigateur Web, la fiche est affichée et exécutée comme application incorporée dans le navigateur. C'est-à-dire qu'elle s'exécute dans le même processus que le navigateur.

Exercices chapitre 8

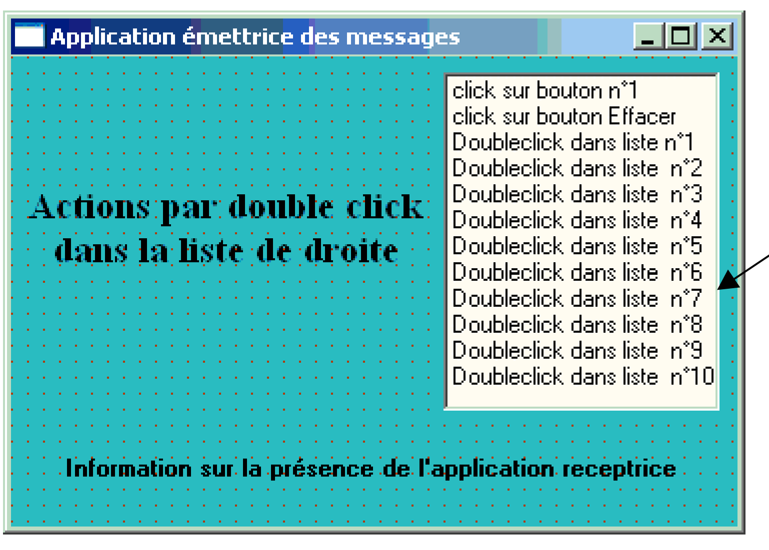
Ex-1 : Exercice entièrement traité sur la prise de contrôle d'une application par une autre à travers des messages systèmes. Le programme comporte une application de reception de messages et une application émettant des messages vers l' application de reception. L'application émettrice envoie des messages qui sont interprétés par la réceptrice comme des manipulations de la souris locale (déplacement, click, double click).




```
unit Unit1;

 interface

 uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

 type
  TFAppliRecept = class(TForm)
  Button1: TButton;
  ListBox1: TListBox;
  Edit1: TEdit;
  Button2: TButton;
  LabelInfo: TLabel;
  LabelVisu: TLabel;
  procedure FormCreate(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure ListBox1DblClick(Sender: TObject);
  procedure ButtonEffacerClick(Sender: TObject);
  procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
   Y: Integer);
 private
 { Déclarations privées }
  procedure MessageUser(var mess:TMsg; var hand:boolean);
 public
 { Déclarations publiques }
end;

 var
  FAppliRecept: TFAppliRecept;

implementation
```

```
{$R *.dfm}

Const
 WM_ClickBouton1 = WM_User; // message envoyé = WM_USER (<=>interprété click bouton n°1)
 WM_ClickEffacer = WM_User+1; // message envoyé = WM_USER+ 1(<=>interprété click bouton effacer)
 wm_dblclickliste = wm_user+2;
 {message envoyé = wm_user+ 2 (<=>interprété double click dans la liste)
                    wparam contient le rang de l'élément sélectionné dans la liste
 }
 wm_mousesurfiche = wm_user+3;
 {message envoyé = wm_user+ 3 (<=>interprété déplacement souris sur la fiche)
                    WParam contiendra la coordonnée X de la souris
                    lparam contiendra la coordonnée y de la souris
 }


 {--- le gestionnaire de traitement du OMessage de l'application ---}

 procedure TFAppliRecept.MessageUser(var mess:TMsg;var hand:boolean);
 { traite le message wm_User intercepté comme un ordre donné à la fenêtre.
   il est traité à partir du niveau onmessage de l'application
 }
 begin
  if mess.message=WM_ClickBouton1 then // message envoyé = WM_USER ici
  begin
   LabelInfo.Caption:='WM_ClickBouton1 : simulant un click sur bouton n°1';
   Button1.Click;
  end
  else
   if mess.message=wm_clickeffacer then  // message envoyé = wm_user+1 ici
   begin
    LabelInfo.Caption:='WM_ClickEffacer : simulant un click sur bouton Effacer';
    Button2.Click;
   end
   else
    if mess.message=wm_dblclickliste then
    begin
     LabelInfo.Caption:='WM_DblClickListe : simulant un double click sur la liste';
     ListBox1.ItemIndex:=mess.wParam; //le rang de l'élément sélectionné
     ListBox1DblClick(ListBox1)
    end
    else
     if mess.message=wm_mousesurfiche then
     begin
      LabelInfo.Caption:='WM_MouseSurFiche : simulant la position de la souris sur la fiche';
      FormMouseMove(self, [ ], mess.wParam, mess.lParam)
     end;
  inherited; //laisse delphi s'occupper des autres messages
 end;

 {------------------ LES GESTIONNAIRES D'EVENMENTS -------------------}

 procedure TFAppliRecept.FormCreate(Sender: TObject);
 //connexion du gestionnaire de OnMessage de l'application
 begin
  Application.OnMessage:=MessageUser;
 end;
```

```
{---- Les gestionnaires d'évènements qui seront appelés par l'émetteur
et qui fonctionnent déjà en local lorsque l'appli est activée ----
}

procedure TFAppliRecept.Button1Click(Sender: TObject);
//gestion du click du button1
begin
 Edit1.text:='Le bouton n° 1 vient d''être clické';
end;

procedure TFAppliRecept.ButtonEffacerClick(Sender: TObject);
//gestion du click du button2
begin
 Edit1.Clear
end;

procedure TFAppliRecept.ListBox1DblClick(Sender: TObject);
//gestion du double click du ListBox1
begin
 with Sender as TListBox do
  Edit1.text:='choix dans la liste : '+ListBox1.Items[itemindex];
end;

procedure TFAppliRecept.FormMouseMove(Sender: TObject; Shift: TShiftState;
 X, Y: Integer);
 //gestion du mousemove sur la fiche
 begin
  Edit1.text:='souris/ x = '+inttostr(x)+' y = '+inttostr(y);
  LabelVisu.Top:=Y;
  LabelVisu.Left:=X;
 end;

end.
```



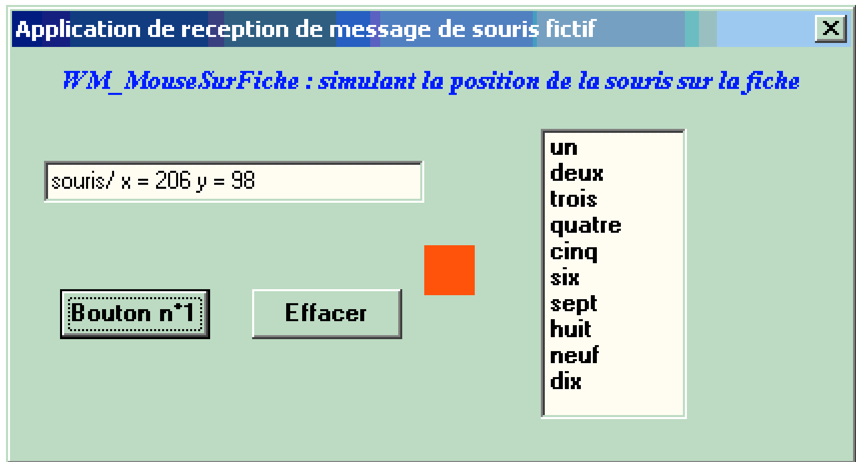
Application émettrice qui envoie les messages comme des ordres donnés à la souris de l'application réceptrice.

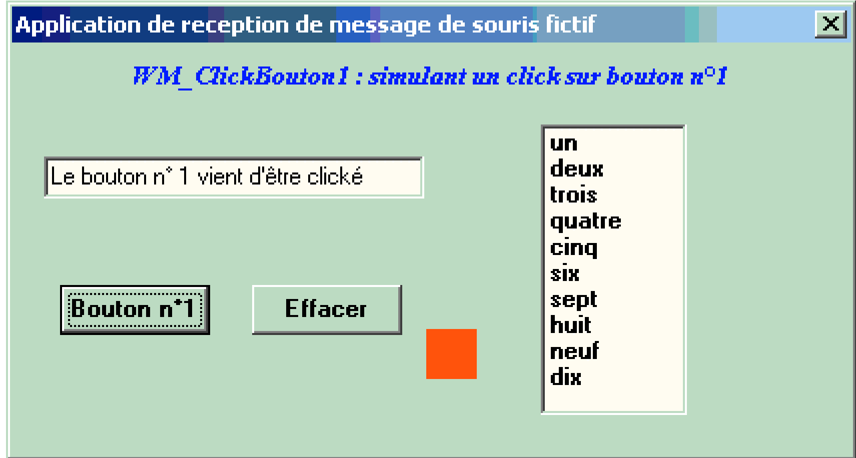
```
unit Unit1;
{    cette application contrôle l'action de la souris dans une autre
application au niveau du click sur un bouton, double clik et mouse move.
```

```
le nom de la classe TForm de la fenêtre de l'appli réceptrice est TFAppliRecept.
On construit 4 messages utilisateurs :
WM_ClickBouton1 = WM_User;
WM_ClickEffacer = WM_User+1;
WM_DblClickListe = WM_User+2;
WM_MouseSurFiche = WM_User+3;

et PostMessage est chargée de les expédier à  TFAppliRecept (avec des paramètres éventuels)

}
interface

 uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls;

 type
  TForm1 = class(TForm)
  LabelInfoMess: TLabel;
  ListBoxMess: TListBox;
  Label1: TLabel;
  procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
   Y: Integer);
  procedure ListBoxMessDblClick(Sender: TObject);
 private
 { Déclarations privées }
 public
 { Déclarations publiques }
end;

 var
  Form1: TForm1;

implementation

 {$R *.dfm}

 procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
 var
  Wnd:HWnd;
  begin
   Wnd:=FindWindow('TFAppliRecept',nil);// recherche de la fenêtre
   if Wnd<>0 then// si instance de la fenêtre  trouvée
   begin
    LabelInfoMess.Caption:='Fenêtre TFAppliRecept trouvée';
    PostMessage(Wnd,WM_USER+3,X,Y);// WM_MouseSurFiche
    {les paramètres X et Y donnent les coord. de la sourisà utiliser
    dans la fenêtre réceptrice
    }
   end
  end;


 procedure TForm1.ListBoxMessDblClick(Sender: TObject);
 var
  Wnd:HWnd;
  begin
   Wnd:=FindWindow('TFAppliRecept',nil);// recherche de la fenêtre
   if Wnd<>0 then// si instance de la fenêtre  trouvée
```

```
 begin
  LabelInfoMess.Caption:='Fenêtre TFAppliRecept trouvée';
  case
   listboxmess.itemindex of
   0:PostMessage(Wnd,WM_USER,0,0); //WM_ClickBouton1
   1:PostMessage(Wnd,WM_USER+1,0,0);// WM_ClickEffacer
   2..11:PostMessage(Wnd,WM_USER+2,ListBoxMess.itemindex-2,0);// WM_DblClickListe
   {la valeur du paramètre ListBoxMess.itemindex-2 indique le rang de
       l'élément à clicker dans la liste de la fenêtre de réception
   }
  end;
 end
end;

end.
```

Exemples

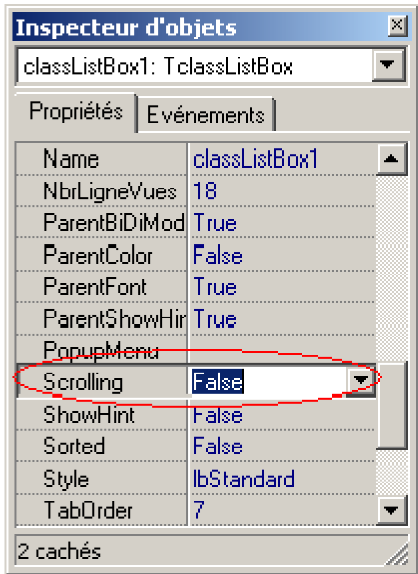


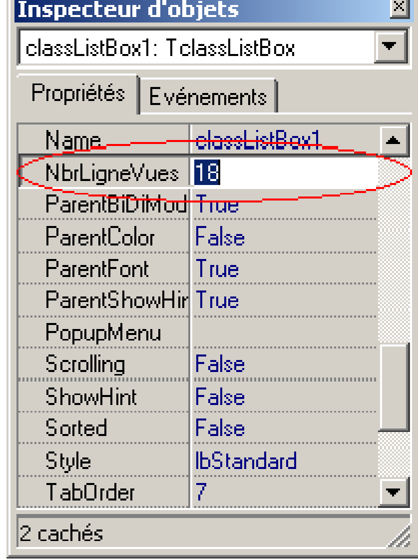


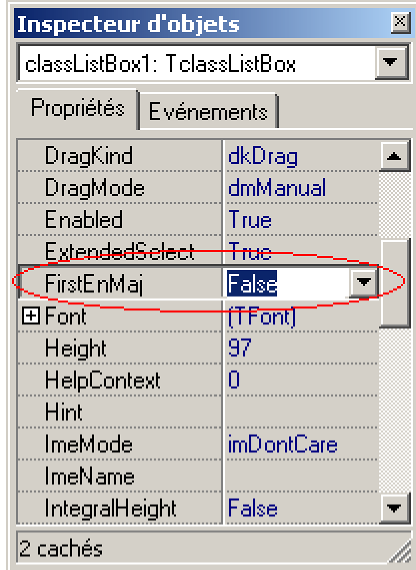
etc…

Ex-2 : Complétez la Unit suivante de telle manière que la classe TclassListBox représente un composant héritant des TListBox avec les améliorations suivantes :

Trois nouvelles propriétés :
Scrolling:boolean *{permettant le défilement automatique du texte}*
NbrLigneVues:integer *{permettant de fixer le nombres de lignes de texte affichées}*
FirstEnMaj:boolean *{permettant d'autoriser la mise en majuscule du premier caractère de tous les lignes de la liste}*

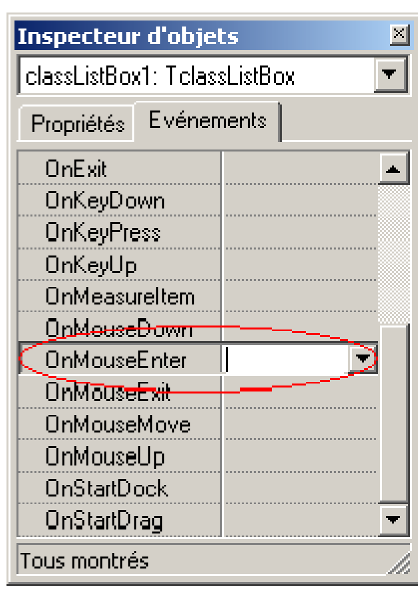


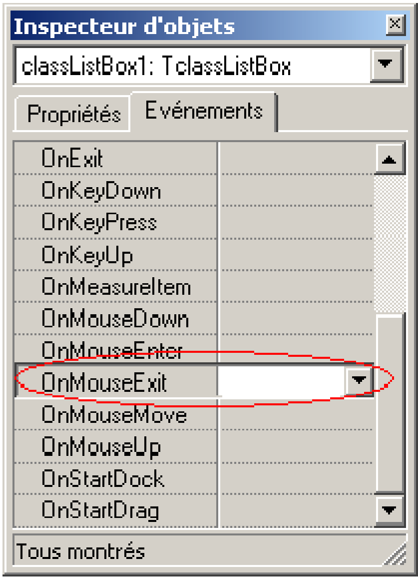




Deux nouveaux événements :

OnMouseEnter: *{interceptant l'entrée de la souris dans le composant}*
OnMouseExit: *{interceptant la sortie de la souris hors du composant}*





Code fourni : Squelette de la Unit à compléter

```
unit UclassListBox;

interface
uses stdctrls,Messages, Controls,Windows,Classes,Sysutils,extctrls;

type

 TclassListBox=class(TListBox)
  private
   FOnMouseEnter: TNotifyEvent;
   FOnMouseExit: TNotifyEvent;
   FFirstEnMaj:boolean;// champ d'autorisation de mise en majuscule du 1er caractère
   FNbrLigneVues:integer;// champ indiquant le nombre de lignes affichables dans la liste
   FScrolling:boolean; // champ d'autorisation de défilement automatique
```

```
    Tempo:TTimer;// Timer de défilement ligne à ligne de la liste
    procedure SetMaj(x:boolean);
    procedure CMMOUSEENTER(var mess:TMessage);message CM_MOUSEENTER; // VCL borland
    procedure CMMOUSELEAVE(var mess:TMessage);message CM_MOUSELEAVE ; // VCL borland
    procedure LBADDSTRING(var mess:TMessage);message  LB_ADDSTRING; //system
    procedure WMSIZE(var mess:TMessage);message WM_SIZE; //system
    procedure setScrolling(x:boolean);
    procedure setNbrLigneVues(x:integer);
    procedure OnTempo(Sender:TObject);
   public
    constructor Create(AOwner: TComponent); override;
   published
    property NbrLigneVues:integer read FNbrLigneVues write setNbrLigneVues;
    property Scrolling:boolean read FScrolling write setScrolling;
    property FirstEnMaj:boolean read FFirstEnMaj write SetMaj;
    property OnMouseEnter: TNotifyEvent read FOnMouseEnter write FOnMouseEnter;
    property OnMouseExit: TNotifyEvent read FOnMouseExit write FOnMouseExit;

 end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents( 'Exemples', [TclassListBox]);
end;

{ TclassListBox }
constructor TclassListBox.Create(AOwner: TComponent);
//le constructeur du composant
begin

end;


procedure TclassListBox.WMSIZE(var mess: TMessage);
{lors du changement de taille du composant :
   - le nombre de lignes vues change,
   - la première ligne du texte doit être affichée au début
   - le défilement est éventuellement arrêté selon le nombre de lignes affichées
}
begin

end;

{----------- entrée et sortie de la souris -----------------}
procedure TclassListBox.CMMOUSEENTER(var mess: TMessage);
//lance le gestionnaire d'événement OnmouseEnter
begin

end;

procedure TclassListBox.CMMOUSELEAVE(var mess: TMessage);
//lance le gestionnaire d'événement OnmouseExit
begin

end;
```

```
{----------- le premier caractère d'une ligne  -----------------}
procedure TclassListBox.LBADDSTRING(var mess: TMessage);
{lorsqu'une nouvelle ligne est insérée dans la liste, le premier caractère est mis en majuscule
 (si le composant est autorisé à mettre en majuscule).}
begin

end;

procedure TclassListBox.SetMaj(x: boolean);
{Met en majuscule tous les premiers caractères de chaque ligne si x est true,  sinon met tous
ces premiers caractères en minuscule et positionne FirstEnMaj:boolean  à la valeur adéquate.}
begin

end;

{-----------le défilement automatique -----------------}
procedure TclassListBox.OnTempo(Sender: TObject);
{le défilement ligne à ligne de toute la liste jusqu'à la fin  en boucle si le nombre ligne de la liste est plus grand
que le  nombre de ligne affichables (NbrLigneVues:integer donne ce nombre)
et si le défilementest autorisé (Scrolling:boolean donne cette autorisation)
}
begin

end;

procedure TclassListBox.setScrolling(x: boolean);
{positionne le défilement ligne à ligne de toute la liste jusqu'à la fin
 en boucle si le nombre ligne de la liste est plus grand que le
 nombre de ligne affichables (NbrLigneVues:integer donne ce nombre)
}
begin

end;

procedure TclassListBox.setNbrLigneVues(x: integer);
{positionne le nombre de ligne affichables (NbrLigneVues:integer)
 recalcule et modifie la hauteur du composant en fonction du nombre de lignes
 que l'on veut afficher.
}
begin

end;

end.
```

Messages à utiliser (conseils) :

{
CM_MOUSELEAVE , CM_MOUSEENTER : pour la souris
LB_ADDSTRING , LB_GETTOPINDEX , LB_SETTOPINDEX : pour les ListBox
WM_SIZE : pour le redimensionnement du composant
WM_VSCROLL : pour la barre de défilement vertical
}

2°) Fournissez le composant prêt à installer .

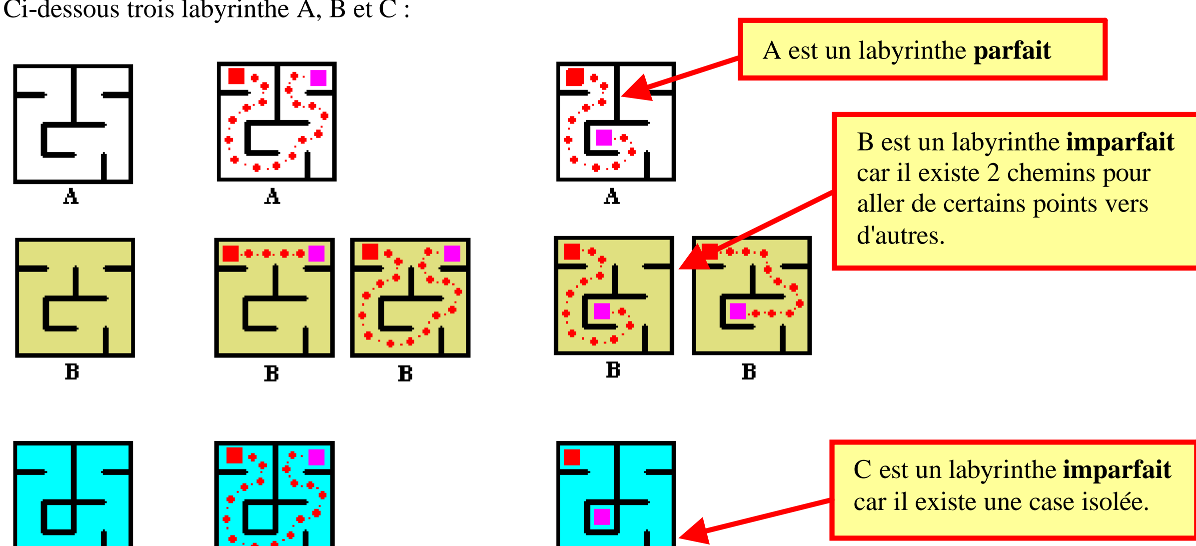
Ecrivez un programme Delphi de test du composant.

Ex-3 : non déterminisme et retour arrière algorithme d'exploration d'un labyrinthe

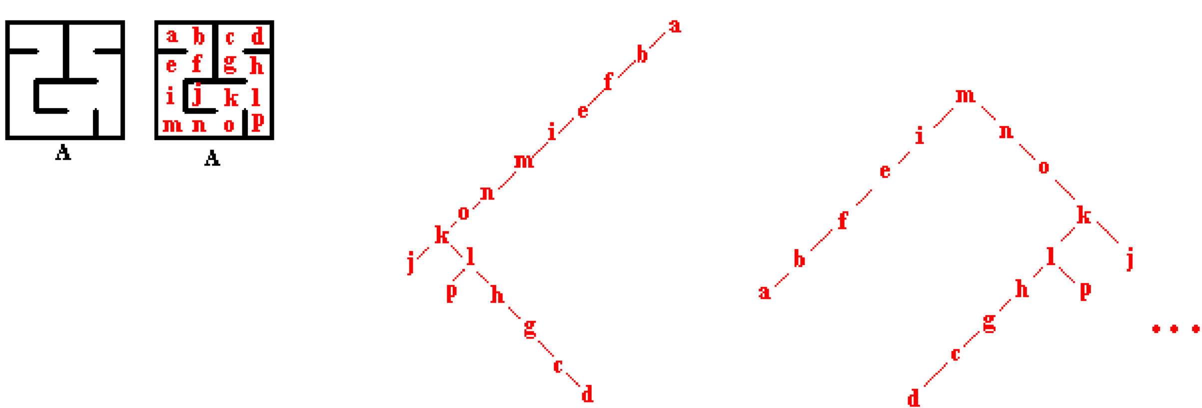
Un labyrinthe est représenté par un tableau de cases, c'est un labyrinthe **parfait** si :

> Deux cases quelconques sont joignables
> par un chemin unique.il n'existe pas de case isolée.

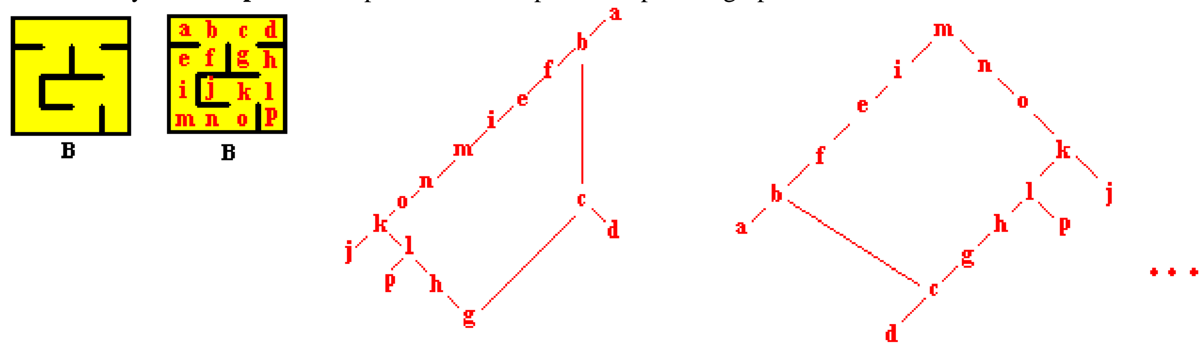
Ci-dessous trois labyrinthe A, B et C :



Les parcours dans un labyrinthe **parfait** peuvent être représentés par des arbres dont chaque noeud a au plus 3 fils :



Dans un labyrinthe **imparfait** les parcours sont représentés par des graphes non arborescents :



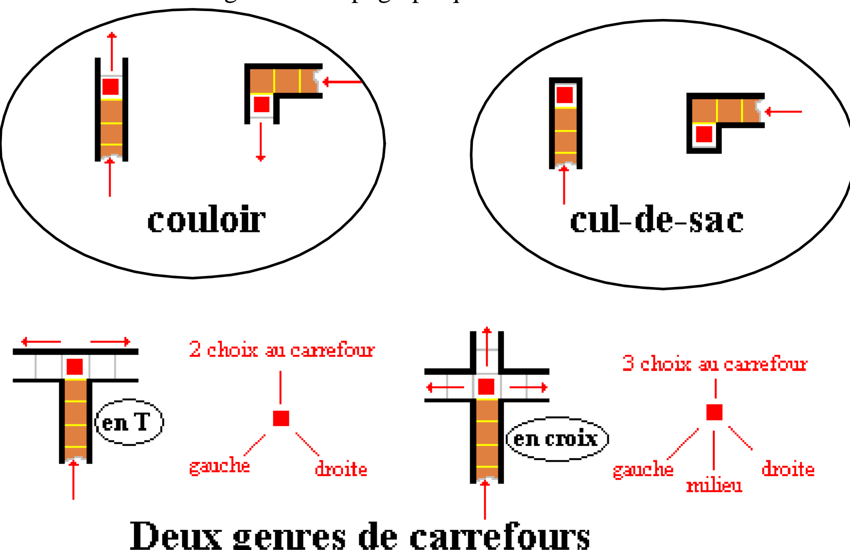
imparfait

Objectif :

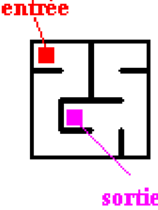
Nous allons faire explorer un labyrinthe semi-parfait (il peut y avoir plusieurs chemins entre deux cases, mais il n'y a aucune case isolée) à un **pélerin,** il aura pour mission de trouver une sortie s'il y en a une, ou bien de nous assurer qu'après exploration complète de ce labyrinthe, il ne comporte pas de sortie.

Pour notre pélerin un labyrinthe à explorer est caractérisé par une **entrée,** une succession de **couloirs**, de **cul-de-sacs** et de **carrefours** (et éventuellement une sortie).

Ci-dessous les configurations topographiques associées :



Dans un labyrinthe parfait, nous sommes assurés que s'il existe une sortie, il existe alors un chemin unique allant de la case marquée **entrée** vers la case marquée **sortie**.



Le travail de notre pélerin va être de chercher ce chemin s'il existe, par exploration exhaustive.

On apprend son "métier" au pélerin

Nous ne voulons pas que notre pélerin se perde dans le labyrinthe en explorant plusieurs fois le même chemin aussi va-t-on lui enseigner l'**art étrange** du parcours en profondeur avec retour arrière (back-tracking disent les anglophones) sans repasser sur un chemin déjà exploré. Notre centre de formation de pélerin explorateur de labyrinthe ne reculant devant aucune dépense, munit notre voyageur d'une lampe torche à durée de vie importante, de deux sacs chacun remplis de cailloux, l'un de cailloux roses l'autre de cailloux gris.

L'instructeur lui apprend qu'il va suivre des couloirs dans le labyrinthe, qu'il va changer de direction à des carrefours pour emprunter un nouveau couloir, qu'il va aussi tomber sur des culs-de-sacs. On lui apprend qu'il va devoir changer son comportement dès qu'il rencontre un cul-de-sac.

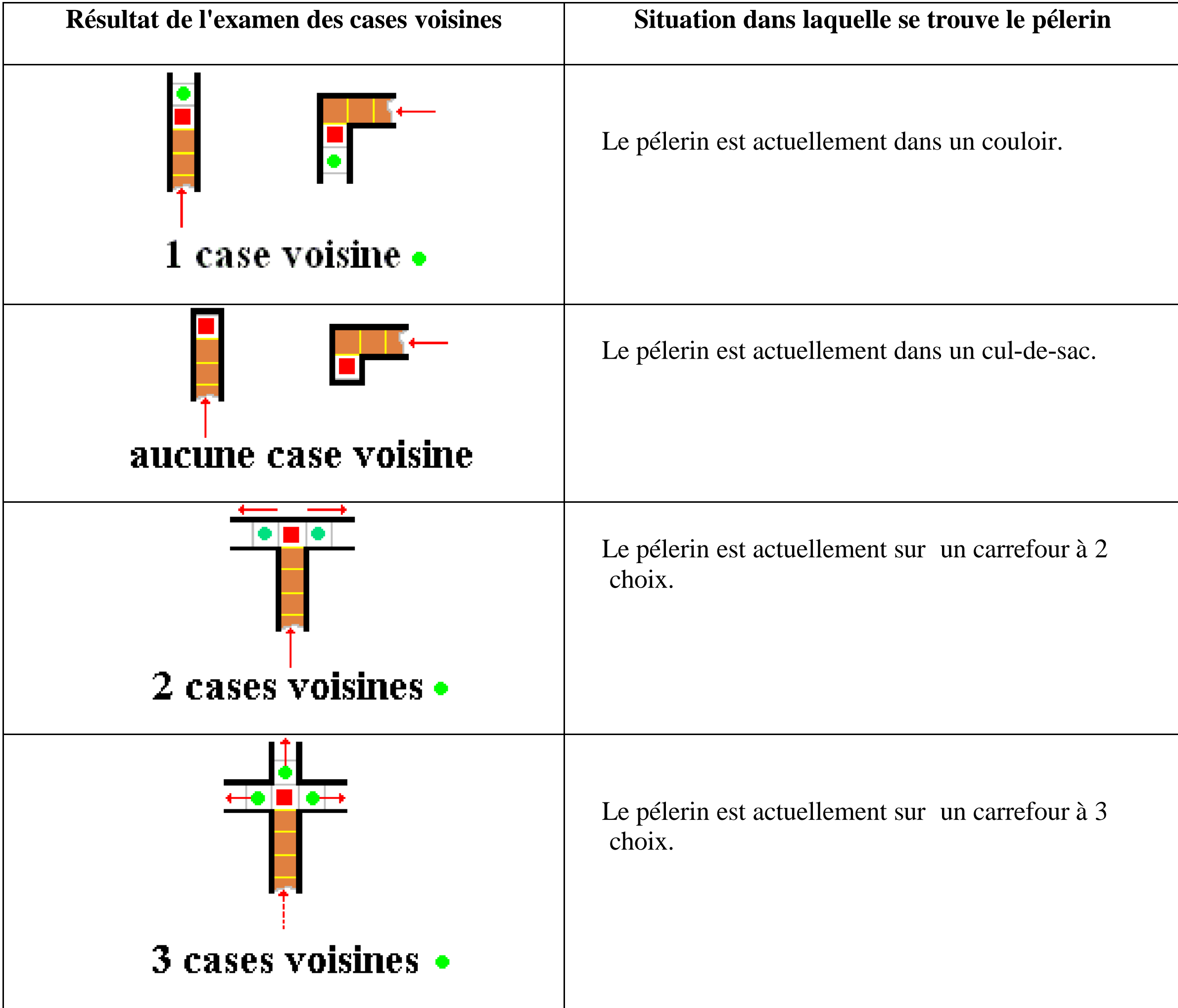
Tantqu'il chemine dans un couloir on lui conseille de déposer derrière lui un caillou rose, s'il arrive à un carrefour il dépose toujours un caillou rose et emprunte un des couloirs disponibles.

Dès qu'il rencontre un cul-de-sac, il doit évidemment revenir sur ses pas en ramassant les cailloux roses qu'il

a semés, lorqu'il arrive en revenant sur ses pas, à un carrefour, il dépose derrière lui un caillou gris à l'entrée du couloir qu'il est en train de quitter. Il doit aller au milieu du carrefour (qui contient un caillou rose) et il examine le seuils des différents couloirs qui débouchent sur ce carrefour :

- Un seul couloir a un caillou rose sur son seuil, les autres couloirs qu'il a déjà explorés ont un caillou gris sur leur seuil, ceux qu'il n'a pas encore explorés n'ont aucun caillou sur leur seuil.
- Dans l'éventualité où tous les seuils ont un caillou, il doit impérativement ramasser le caillou rose du carrefour et emprunter l'unique couloir dont le seuil est marqué par un caillou rose, tout en ramassant les cailloux rose semés dans ce couloir (il vient en fait d'explorer tous les chemins qui partaient de ce carrefour)

L'examen des seuils doit se faire par observation des cases (des entrées de couloir), le pélerin doit connaître le nombre de cases voisines de celle sur laquelle il se trouve, il en déduira le type de situation dans laquelle il est :



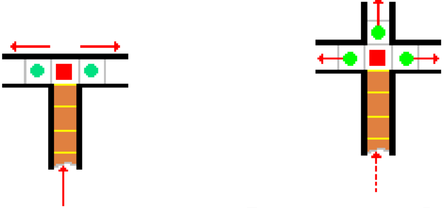
Résultat de l'examen des cases voisines	Situation dans laquelle se trouve le pélerin
1 case voisine	Le pélerin est actuellement dans un couloir.
aucune case voisine	Le pélerin est actuellement dans un cul-de-sac.
2 cases voisines	Le pélerin est actuellement sur un carrefour à 2 choix.
3 cases voisines	Le pélerin est actuellement sur un carrefour à 3 choix.

Bien entendu on a indiqué au pélerin que dès qu'il trouvait une sortie, il devait l'emprunter et son voyage se terminait là.

Algorithme de parcours en profondeur

Nous proposons une version itérative et une version récursive du parcours dans le labyrinthe.

```
Algorithme  Labyrinthe
{ Un algorithme de déplacement dans  un labyrinthe parfait. Version ITERATIVE  }
global
  le labyrinthe
  sens  € Booleen  // indique le sens actuel de parcours
  SortieFound  € Booleen  // une sortie a été trouvée
utilise
  DeplacerAller  // déplacement d'une case dans le sens exploration
  DeplacerEnRetour  // déplacement d'une case en revenant sur ses pas

  debut
  tantque  non  SortieFound   faire
   si  sens = aller  alors
     DeplacerAller
    sinon
     DeplacerEnRetour
    fsi
  ftant
 fin  //   Labyrinthe
Algorithme  DeplacerAller
{ Cheminement en mode "aller" à travers des couloirs et des carrefours : Actions
  lorsque le pélerin est sur une case du  labyrinthe.
}
global
  le labyrinthe
 sens  € Booleen  // indique le sens actuel de parcours
 SortieFound  € Booleen  // une sortie a été trouvée

  debut
  si  la case est une  sortie alors
    SortieFound  <-- vrai ;
    arrêt de l'exploration
   sinon
    Examen des cases voisines ;
   si  aucune case voisine  alors  //cul-de-sac

    aucune case voisine
    { il change de sens et se prépare è revenir sur ses pas }
     sens  <--  retour ;
     le pélerin ramasse le caillou rose ;
   sinon
    si  une seule case est voisine  alors  //couloir

     1 case voisine
     le pélerin se déplace  vers  cette case ;
     si  la case actuelle n'est pas l' Entrée alors
       il laisse un caillou rose sur la case actuelle ;
     fsi
    sinon  //carrefour (2 ou 3 voisins)
```

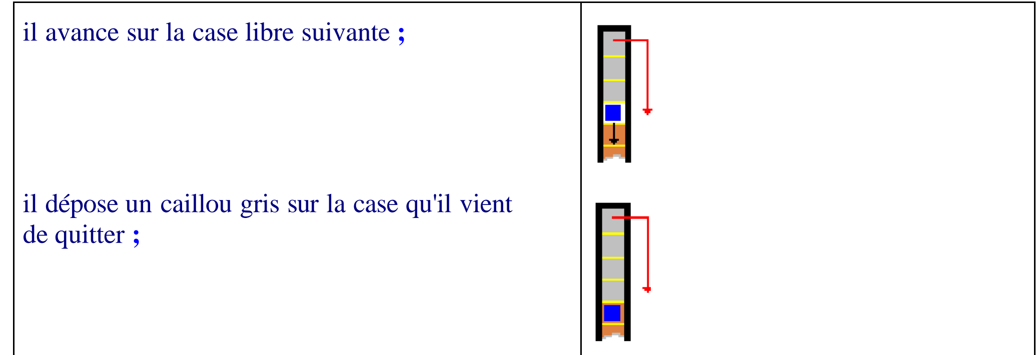


2 cases voisines , **3 cases voisines**

le pélerin choisit aléatoirement une case libre **;**
le pélerin se déplace **vers** cette case **;**
si la case actuelle n'est pas l' **Entrée alors**
il laisse un caillou rose sur la case actuelle **;**
fsi
fsi
fsi
fin *// DeplacerAller*

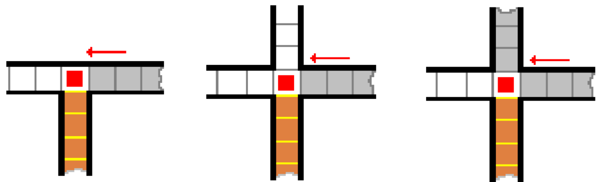
Algorithme DeplacerRetour
{ Cheminement en mode "retour" à travers des couloirs et des carrefours : Actions lorsque le pélerin est sur une case du labyrinthe. }
global
le labyrinthe
sens € Booleen *// indique le sens actuel de parcours*

debut
si le pélerin est dans un couloir **alors**

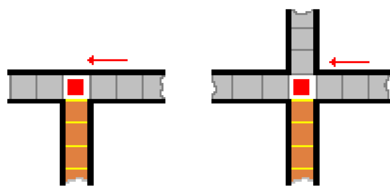


il avance sur la case libre suivante **;**	
il dépose un caillou gris sur la case qu'il vient de quitter **;**	

sinon *// la case est un carrefour*
évaluation du nombre de case voisines libres **;**
si au moins une case est libre **alors**



le pélerin avance sur cette case libre
sinon *// toutes les cases voisines ont été explorées*
le pélerin retourne en empruntant la case qui contient le caillou rose *// l'entrée du couloir de retour*



fsi
fsi

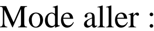
```
fin  // DeplacerRetour
```

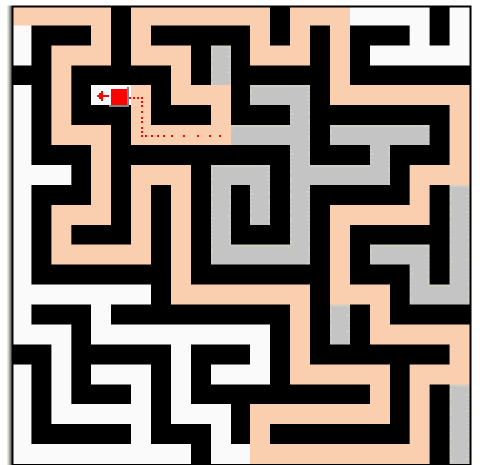
Version récursive

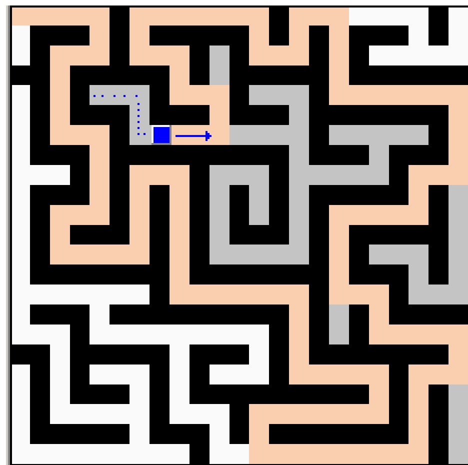
Nous donnons une version récursive qui reprend exactement la stratégie précédente, en particulier les deux algorithmes DeplacerAller et DeplacerEnRetour, en changeant l'itération **tantque** en récursivité (comparez les deux solutions) :

```
Algorithme  Labyrinthe
{ Un algorithme de déplacement dans un labyrinthe parfait. Version RECURSIVE }
global
  le labyrinthe
  SortieFound  € Booleen  // une sortie a été trouvée
  debut
    si  FSortieFound = faux  alors
      seDeplacer
    sinon
      ecrire('La sortie a déjà été trouvée !')
    fsi
 fin  //   Labyrinthe

Algorithme  SeDeplacer
{ le bloc  RECURSIF }
global
  le labyrinthe
  SortieFound  € Booleen  // une sortie a été trouvée
utilise
  DeplacerAller  // déplacement d'une case dans le sens exploration
  DeplacerEnRetour  // déplacement d'une case en revenant sur ses pas
 debut
   si  sens = aller  alors
     DeplacerAller
   sinon
     DeplacerEnRetour
   fsi ;
   SeDeplacer
 fin  //   SeDeplacer
```

Implanter en Delphi les deux versions de cet algorithme. Ci-dessous l'exécution d'un programme Delphi gérant d'une manière animée, le parcours d'un pèlerin (un carré) dans un labyrinthe.

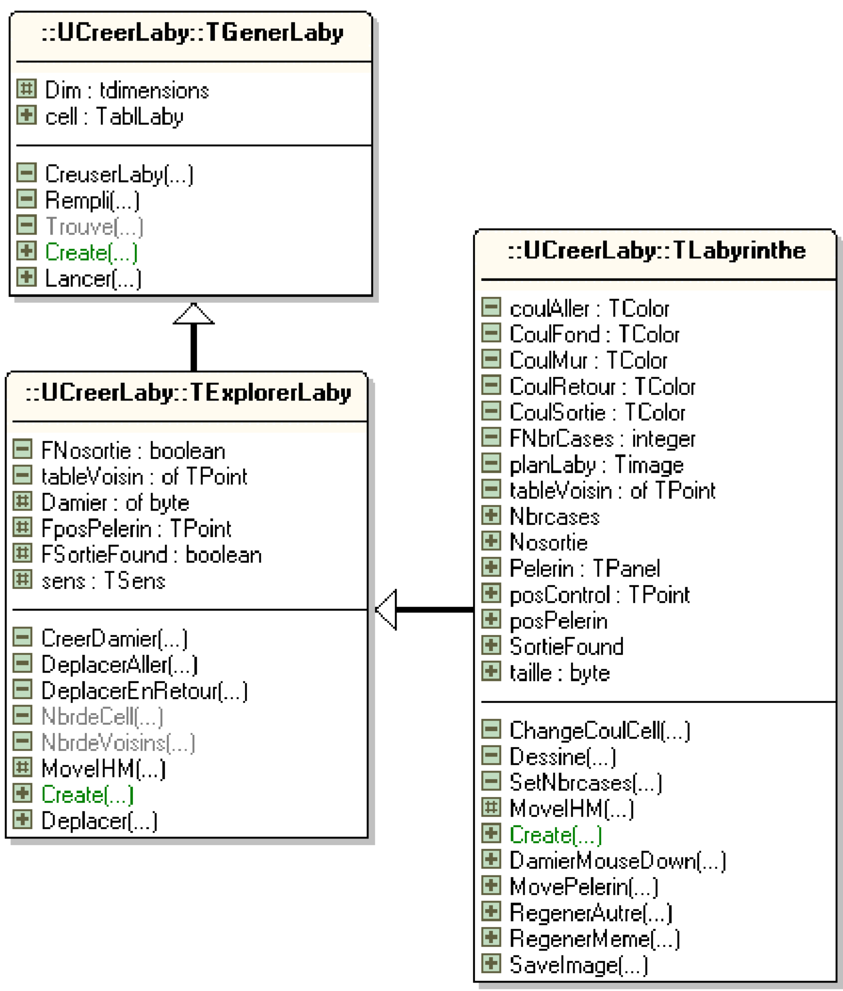






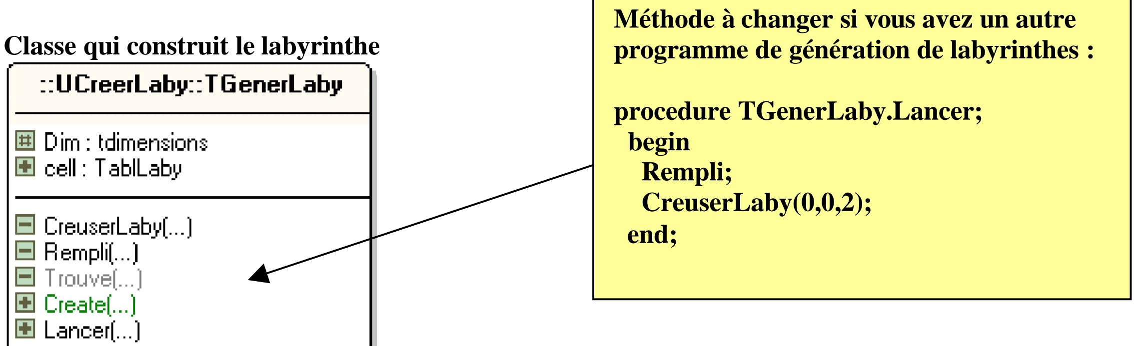
Ex-3 Code solution pratique : Labyrinthe

Une solution d'animation d'un parcours dans un labyrinthe sous forme de 3 classes implémentant un explorateur et un labyrinthe :

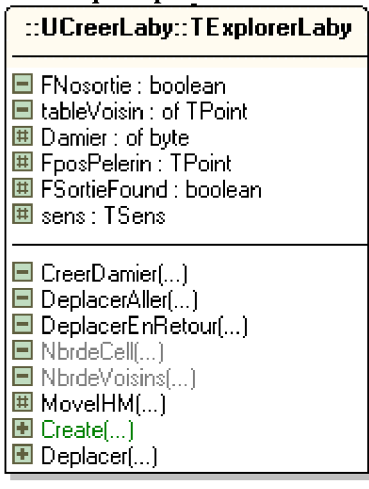


L'algorithme de création aléatoire d'un labyrinthe a été trouvé sur Internet (R.Jolivet), plusieurs autres algorithmes sont proposés, le lecteur peut changer d'algorithme pour obtenir des labyrithes de topologies différentes, il lui suffit de remplacer la méthode Lancer de la classe TGenerLaby.

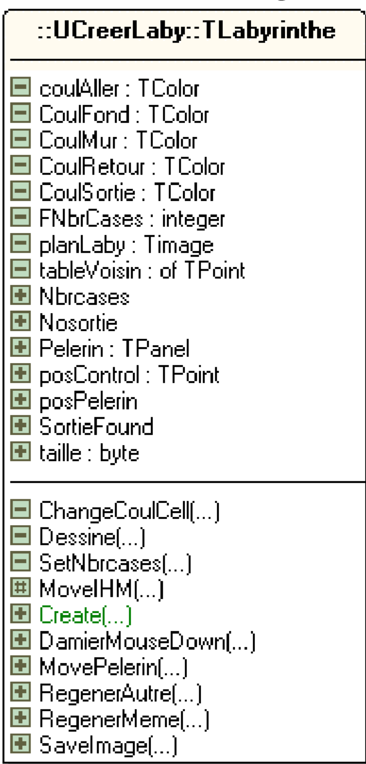
Classe qui construit le labyrinthe



Classe qui explore une case du labyrinthe



Classe dédiée à l'affichage visuel



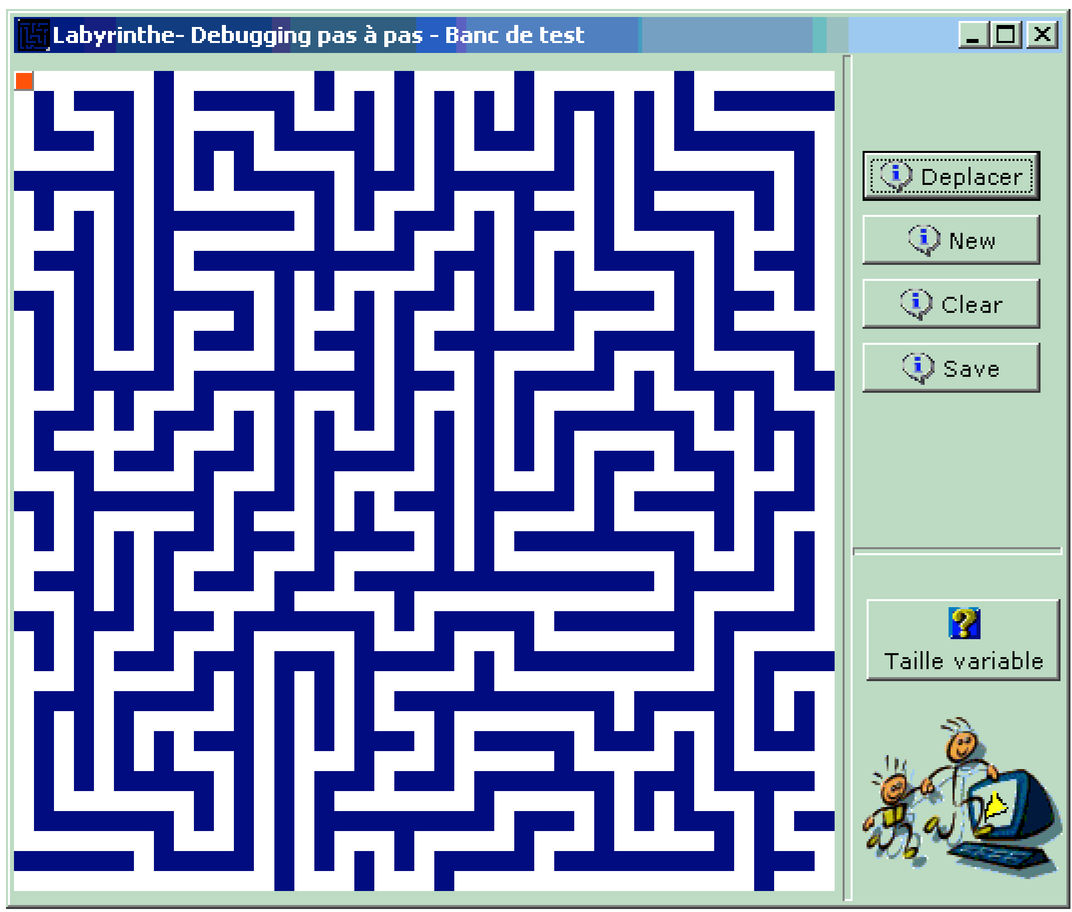
Le programme engendre un labyrinthe aléatoire, puis lance un TTimer qui fait explorer les cases successivement selon l'algorithme de l'énoncé (parcours en profondeur) et affiche visuellement les différents trajets. Le pélerin est un carré de couleur rouge, les cailloux roses sont représentés par une case colorée en rose, les murs sont noirs, les cases libres sont blanches, les cailloux gris sont représentés par une case colorée en gris. Pour bien indiquer visuellement le sens de parcours, le pélerin est de couleur rouge lorsqu'il est en mode aller dans un couloir et en bleu lorsqu'il est en mode retour.

Ci-après une IHM de test du labyrinthe pas à pas

```
unit UFLaby;

 interface

 uses
  UCreerLaby,unit2,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, ComCtrls, StdCtrls, Buttons, Spin, Menus;

 type
  TForm1 = class(TForm)
  Couleur: TColorDialog;
  SaveDialog1: TSaveDialog;
  Image1: TImage;
  PanelPet: TPanel;
  BitBtnPerso: TBitBtn;
  BitBtnNew: TBitBtn;
  BitBtnClear: TBitBtn;
  BitBtn2: TBitBtn;
  BitBtnsave: TBitBtn;
  Bevel1: TBevel;
```

```
 Bevel2: TBevel;
 Image2: TImage;
 procedure FormCreate(Sender: TObject);
 procedure BitBtnPersoClick(Sender: TObject);
 procedure BitBtnNewClick(Sender: TObject);
 procedure BitBtnClearClick(Sender: TObject);
 procedure BitBtn2Click(Sender: TObject);
 procedure BitBtnsaveClick(Sender: TObject);
public
 { Déclarations publiques }
LabyRMD:TLabyrinthe;
end;

var  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
 LabyRMD:=TLabyrinthe.Create(Image1);
 PanelPet.Width:= LabyRMD.taille;
 PanelPet.Height:= LabyRMD.taille;
 LabyRMD.Pelerin:=PanelPet;
 LabyRMD.posPelerin:=Point(0,0);
 LabyRMD.MovePelerin(Point(0,0));
end;


procedure TForm1.BitBtnPersoClick(Sender: TObject);
begin
 LabyRMD.Deplacer
end;

procedure TForm1.BitBtnNewClick(Sender: TObject);
begin
 LabyRMD.RegenerAutre
end;

procedure TForm1.BitBtnClearClick(Sender: TObject);
begin
 LabyRMD.RegenerMeme
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
 form2.show
end;

procedure TForm1.BitBtnsaveClick(Sender: TObject);
begin
 LabyRMD.SaveImage;
end;

end.
```


Aspect général de l'IHM de test du Labyrinthe :




```
unit UCreerLaby;
{ un design pattern sur les labyrinthes }
interface
 uses Windows,Classes, Sysutils,graphics,extctrls,Controls,Dialogs;
 const
  maxTab=40;
  depart=0;
  carrefour=1;
  libre=2;
  markAller=3;
  markRetour=4;
  mur=10;
  sortie=20;
 type
  tdimensions=
  record
   largeur,hauteur:integer;
  end;
  TablLaby=array[0..maxTab,0..maxTab] of boolean;
  TSens=(aller,retour);


TGenerLaby=class
```

::UCreerLaby::TGenerLaby

Dim : tdimensions
cell : TablLaby

CreuserLaby(...)
Rempli(...)
Trouve(...)
Create(...)
Lancer(...)

```
 private
  procedure Rempli;
  function Trouve(x,y:integer;dir:integer;var surf:TDimensions;var newDir:integer):boolean;
  procedure CreuserLaby(x,y,d:integer);
 protected
  Dim:TDimensions;
 public
  cell:TablLaby;
  procedure Lancer;
  constructor Create(Dim:TDimensions);virtual;
end;

TExplorerLaby=class(TGenerLaby)
```

::UCreerLaby:: TExplorerLaby

FNosortie : boolean
tableVoisin : of TPoint
Damier : of byte
FposPelerin : TPoint
FSortieFound : boolean
sens : TSens

CreerDamier(...)
DeplacerAller(...)
DeplacerEnRetour(...)
NbrdeCell(...)
NbrdeVoisins(...)
MoveIHM(...)
Create(...)
Deplacer(...)

```
 private
  tableVoisin:array[1..4]of TPoint;
  procedure CreerDamier;
  function NbrdeCell(etat:integer):integer;
  function NbrdeVoisins:integer;
  procedure DeplacerAller;
  procedure DeplacerEnRetour;
 protected
  sens:TSens;
  FposPelerin:TPoint;
  FSortieFound,FNosortie:boolean;
  Damier:array[0..maxTab,0..maxTab] of byte;
  procedure MoveIHM(x,y:integer;posExpl:TPoint;caillou:integer);virtual;
 public
  procedure Deplacer;
  constructor Create(Dim:TDimensions);override;
end;
```

TLabyrinthe=class(TExplorerLaby)

::UCreerLaby::TLabyrinthe

coulAller : TColor
CoulFond : TColor
CoulMur : TColor
CoulRetour : TColor
CoulSortie : TColor
FNbrCases : integer
planLaby : Timage
tableVoisin : of TPoint
Nbrcases
Nosortie
Pelerin : TPanel
posControl : TPoint
posPelerin
SortieFound
taille : byte

ChangeCoulCell(...)
Dessine( )
SetNbrcases(...)
MoveIHM(...)
Create(...)
DamierMouseDown(...)
MovePelerin(...)
RegenerAutre(...)
RegenerMeme(...)
SaveImage(...)

```
 private
  planLaby:Timage;
  tableVoisin:array[1..4]of TPoint;
  FNbrCases:integer;
  CoulMur,CoulFond,coulAller,CoulRetour,CoulSortie:TColor;
  procedure SetNbrcases(x:integer);
  procedure Dessine;
  procedure ChangeCoulCell(coord:TPoint;coul:TColor);
 protected
  procedure MoveIHM(x,y:integer;posExpl:TPoint;caillou:integer);override;
 public
  Pelerin:TPanel;
  taille:byte;
  posControl:TPoint;
  property Nbrcases:integer read FNbrcases write SetNbrcases;
  property SortieFound:boolean read FSortieFound;
  property Nosortie:boolean read FNosortie;
  property posPelerin:TPoint read FposPelerin write FposPelerin;
  procedure DamierMouseDown(Sender: TObject; Button: TMouseButton;
   shift: tshiftstate;
   x, y: integer);
  procedure MovePelerin(vers:TPoint);
  procedure RegenerMeme;
  procedure RegenerAutre;
  procedure SaveImage;
  constructor Create(Visu:TImage);
end ;

implementation

 { TLabyrinthe
   code Damier
     0 --> départ
     1 --> carrefour
     2 --> libre
```

```
  3 --> aller
  4 --> retour
 10 --> mur
 20 --> sortie

 La fonction récursive de creusement tirée de
 "logiciel pour créer des labyrinthes" de Raphaël Jolivet
 Mail: raphael.jolivet@wanadoo.fr
}


{-------------------------- TGenerLaby ---------------------- }

constructor TGenerLaby.Create(Dim:TDimensions);
begin
 inherited Create;
 self.Dim:=Dim;
 Lancer
end;

procedure TGenerLaby.Lancer;
begin
 Rempli;
 CreuserLaby(0,0,2);
end;

procedure TGenerLaby.CreuserLaby(x, y, d: integer);
var surface:TDimensions;
 dir:integer;
 begin
  dir:=random(1000)+1;
  while trouve(x,y,d,surface,dir)do
  begin
   cell[surface.largeur,surface.hauteur]:=true;
   cell[round((surface.largeur+x)/2),round((surface.hauteur+y)/2)]:=true;
   CreuserLaby(surface.largeur,surface.hauteur,dir);
  end;
 end;

procedure TGenerLaby.Rempli;
var i,j:integer;
begin
 for i:=0 to Dim.largeur do
  for j:=0 to Dim.hauteur do
   cell[i,j]:=false;
end;

function TGenerLaby.Trouve(x, y, dir: integer; var surf: TDimensions; var newDir: integer): boolean;
var surftemp:array[1..4] of TDimensions;
 d:array[1..4] of integer;
 i,k,h:integer;
 p:TDimensions;
 begin
  k:=0;
  for i:=0 to 3 do
  begin
   p.largeur:=x+2*round(cos(Pi/2*(i+dir) ) );
   p.hauteur:=y+2*round(sin(Pi/2*(i+dir) ) );
   if (p.largeur>=0) and (p.hauteur>=0) and (p.largeur<=Dim.largeur)
    and (p.hauteur<=Dim.hauteur)  then
```

```
  if not(cell[p.largeur,p.hauteur]) then
  begin
   k:=k+1;
   surftemp[k]:=p;
   d[k]:=i+dir;
  end;
 end;
 if k>0 then
 begin
  h:=1+random(k);
  surf:=surftemp[h];
  newDir:=d[h];
  result:=true;
 end
 else
  result:=false;
end;

{ ---------------------- TExplorerLaby -----------------------}
constructor TExplorerLaby.Create(Dim:TDimensions);
begin
 inherited;
 CreerDamier;
end;

procedure TExplorerLaby.CreerDamier;
var i,j:integer;
begin
 for i:=0 to Dim.largeur do
  for j:=0 to Dim.hauteur do
   if cell[i,j] then
    Damier[i,j]:=libre
   else
    Damier[i,j]:=mur ;
 Damier[0,0]:=depart;// point de départ
end;

procedure TExplorerLaby.Deplacer;
begin
 if not FSortieFound then
  if sens=aller then
   DeplacerAller
  else
   DeplacerEnRetour
 else
  MessageDlg('La sortie a déjà été trouvée !', mtWarning,[mbOk], 0);
end;

procedure TExplorerLaby.MoveIHM(x,y:integer;posExpl:TPoint;caillou:integer);
begin
 Damier[x,y]:= caillou;
end;

procedure TExplorerLaby.DeplacerAller;
var
 compteVoisins,ptr,i,j,NbrSortie:integer;
 begin
  i:=FposPelerin.x;
  j:=FposPelerin.y;
  NbrSortie:=NbrdeCell(sortie);
```

```
 if NbrSortie>0 then //c'est une sortie
 begin
  FposPelerin:=tableVoisin[1];
  Beep;
  FSortieFound:=true;
  MessageDlg('J''ai trouvé une sortie dans ce labyrinthe !', mtInformation ,[mbOk], 0);
  MoveIHM(i,j,FposPelerin,sortie);
  exit
 end
 else // ce n'est pas une sortie
 begin
  compteVoisins:=NbrdeVoisins;
  case
   comptevoisins of
   0:
   begin // cul-de-sac
    sens:=retour;
    MoveIHM(i,j,Point(i,j),markRetour);
   end;
   1:
   begin // couloir
    FposPelerin:=tableVoisin[1];
    if Damier[i,j]<>depart then
     MoveIHM(i,j,FposPelerin,markAller) //ce n'est pas l'entrée
    else // c'est l'entrée
     MoveIHM(i,j,FposPelerin,depart);
   end;
   2,3:
   begin // carrefour
    ptr:=random(compteVoisins)+1;
    FposPelerin:=tableVoisin[ptr];
    if Damier[i,j]<>depart then //ce n'est pas l'entrée
     MoveIHM(i,j,FposPelerin,carrefour)
    else // c'est l'entrée
     MoveIHM(i,j,FposPelerin,depart);
   end;
  end;
 end
end;

procedure TExplorerLaby.DeplacerEnRetour;
var
 NbrDepart,NewChemin,NbrCarrefour,NbrRetourPoss,i,j:integer;
 begin
  i:=FposPelerin.x;
  j:=FposPelerin.y;
  if (Damier[i,j]<>carrefour)and(Damier[i,j]<>depart) then
  begin
   NbrDepart:=NbrdeCell(depart);
   NbrRetourPoss:=NbrdeCell(markAller);
   NbrdeCell(carrefour);
   FposPelerin:=tableVoisin[1];
   if (NbrRetourPoss=0)and(NbrDepart=1) then //on va au départ
   begin
    NbrdeCell(depart);
    FposPelerin:=tableVoisin[1];
   end;
   MoveIHM(i,j,FposPelerin,markRetour);
  end
  else // on est revenu à un carrefour (point de départ inclu)
```

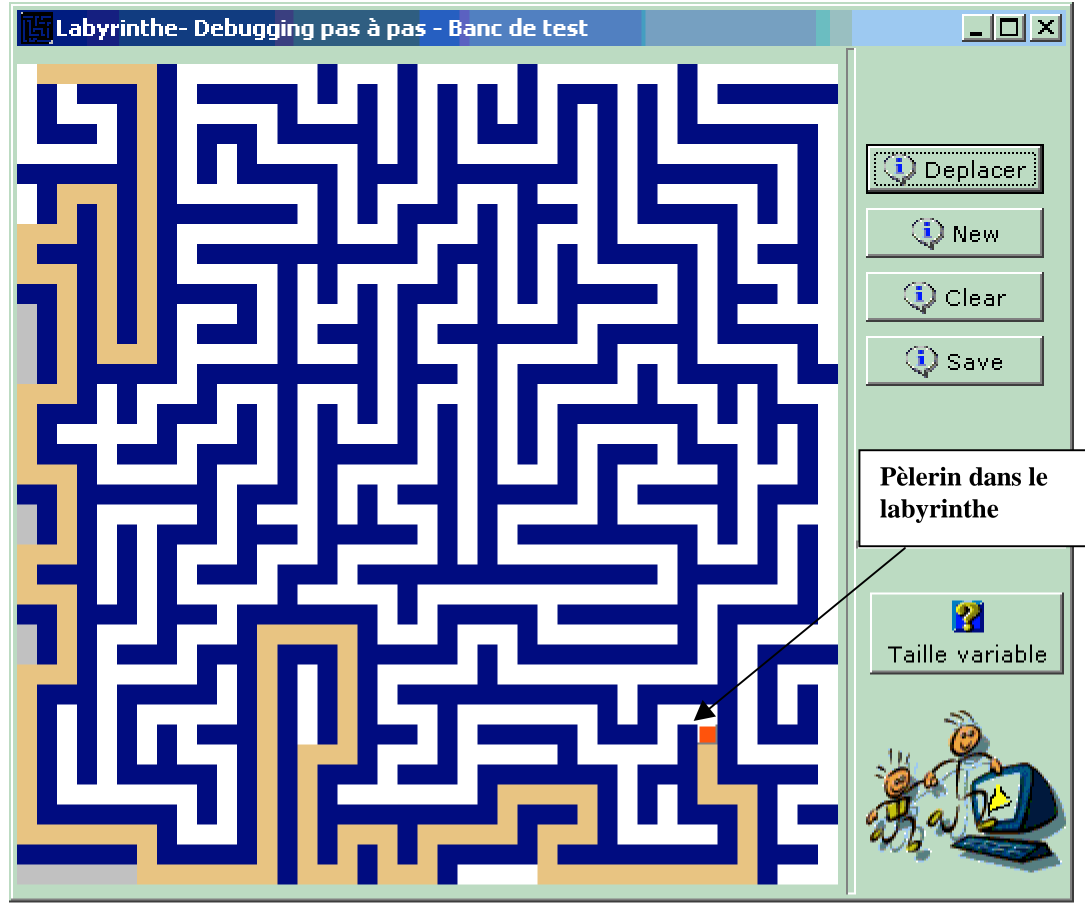
```
 begin
  //-- on est sur un vrai carrefour
  NewChemin:=NbrdeCell(libre);
  if NewChemin>0 then //au moins un chemin encore non exploré
  begin
   if Damier[i,j]<>depart then
    MoveIHM(i,j,FposPelerin,markAller)
   else
    MoveIHM(i,j,FposPelerin,depart);
   sens:=aller;
  end
  else // plus de chemin à explorer
  begin
   NbrRetourPoss:=NbrdeCell(markAller); // il n'y en a qu'une
   {-- tableVoisin[1] contient les coordonnées  de cette cellule (de genre markaller)      }
   if (NbrRetourPoss=0)and(Damier[i,j]=depart) then //on est sur la case départ
   begin
    FNosortie:=true;
    MessageDlg('Il n''y a pas de sortie dans ce labyrinthe !', mtWarning,[mbOk], 0);
    exit
   end;
   FposPelerin:=tableVoisin[1];
   MoveIHM(i,j,FposPelerin,markRetour);
  end
 end;
end;

function TExplorerLaby.NbrdeCell(etat: integer): integer;
var compte:integer;
begin
 compte:=0;
 if FposPelerin.y>0 then
  if Damier[FposPelerin.x,FposPelerin.y-1]=etat then
  begin
   compte:=compte+1;
   tableVoisin[compte]:=Point(FposPelerin.x,FposPelerin.y-1);
  end;
 if FposPelerin.x<Dim.largeur  then
  if Damier[FposPelerin.x+1,FposPelerin.y]=etat then
  begin
   compte:=compte+1;
   tableVoisin[compte]:=Point(FposPelerin.x+1,FposPelerin.y);
  end;
 if FposPelerin.y<Dim.hauteur  then
  if Damier[FposPelerin.x,FposPelerin.y+1]=etat then
  begin
   compte:=compte+1;
   tableVoisin[compte]:=Point(FposPelerin.x,FposPelerin.y+1);
  end;
 if FposPelerin.x>0 then
  if Damier[FposPelerin.x-1,FposPelerin.y]=etat then
  begin
   compte:=compte+1;
   tableVoisin[compte]:=Point(FposPelerin.x-1,FposPelerin.y);
  end;
 result:=compte
end;

function TExplorerLaby.NbrdeVoisins: integer;
begin
```

```
 result:=NbrdeCell(libre)
end;

{ ------------------------  TLabyrinthe --------------------------}
constructor TLabyrinthe.Create(Visu:TImage);
begin
 if not Assigned(Visu) then
 begin
  MessageDlg('Il faut utiliser un TImage pour afficher le plan !', mtError ,[mbOk], 0);
  Exit;
 end;
 planLaby:=Visu;
 FNbrCases:=maxTab;
 Dim.largeur:=FNbrCases;
 Dim.hauteur:=FNbrCases;
 inherited Create(Dim);
 taille:=10;
 sens:=aller;
 FSortieFound:=false;
 FNosortie:=false;
 CoulMur:=clNavy;
 CoulFond:=clWhite;
 coulAller:=$00ABCEFE;
 CoulRetour:=clsilver;//$00E1FEFF;
 CoulSortie:=clFuchsia;
 Randomize;
 planLaby.width:=(Dim.largeur+1)*taille;
 planLaby.height:=(Dim.hauteur+1)*taille;
 planLaby.OnMouseDown:=DamierMouseDown;
 posControl:=Point(planLaby.left,planLaby.top);
 Dessine;
end;

procedure TLabyrinthe.DamierMouseDown(Sender: TObject;  button: tmousebutton;
 shift: tshiftstate;   x, y: integer);
var i,j:integer;
begin
 i:=x div taille;
 j:=y div taille;
 if damier[j,i]=mur then
  exit;
 Damier[j,i]:=sortie;
 TImage(Sender).canvas.brush.color:=CoulSortie;
 TImage(Sender).canvas.fillrect(rect(i*taille,j*taille,(i+1)*taille,(j+1)*taille));
end;

procedure TLabyrinthe.Dessine;
var i,j,Larg,Long:integer;
begin
 for i:=0 to Dim.largeur do
  for j:=0 to Dim.hauteur do
  begin
   if cell[j,i] then
   begin
    planLaby.canvas.brush.color:=CoulFond;
   end
   else
   begin
    planLaby.canvas.brush.color:=CoulMur
   end;
```

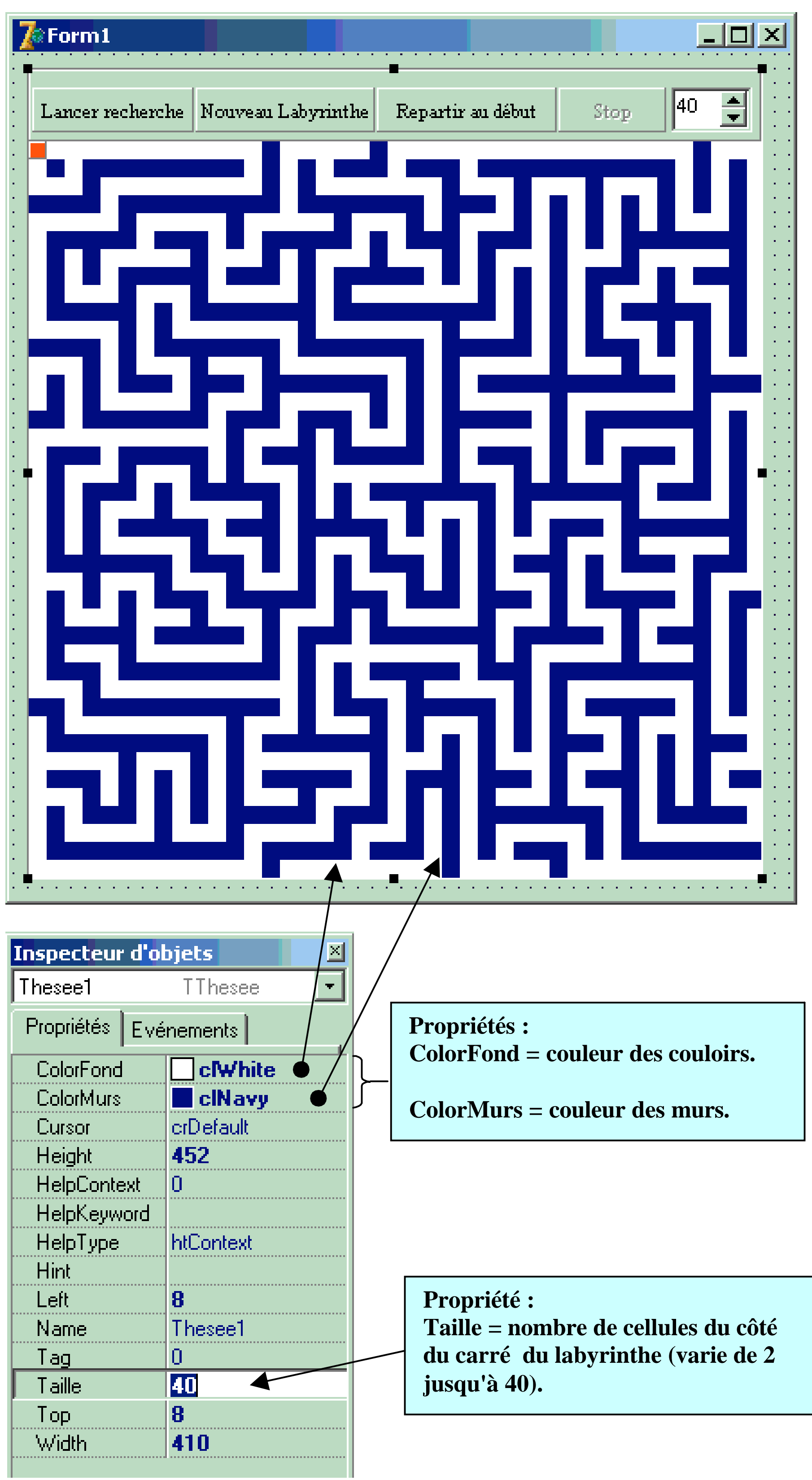
```
  planLaby.canvas.fillrect(rect(i*taille,j*taille,(i+1)*taille,(j+1)*taille));
  end ;
 if Assigned(planLaby) then
 begin
  Larg:= (Dim.largeur+1)*taille+1;
  Long:= (Dim.hauteur+1)*taille+1;
  planLaby.canvas.Pen.Color:=clBlack;
  planLaby.canvas.Pen.Width:=2;
  planLaby.canvas.MoveTo(0,Larg);
  planLaby.canvas.LineTo(Long,Larg);
  planLaby.canvas.LineTo(Long,0);
 end
end;

procedure TLabyrinthe.RegenerAutre;
begin
 Lancer;
 RegenerMeme;
end;

procedure TLabyrinthe.RegenerMeme;
begin
 planLaby.canvas.brush.color:=clSilver;
 planLaby.canvas.fillrect(rect(0,0,planLaby.width,planLaby.height));
 CreerDamier;
 Dessine;
 posPelerin:=Point(0,0);
 MovePelerin(Point(0,0));
 sens:=aller;
 FSortieFound:=false;
 FNosortie:=false;
end;

procedure TLabyrinthe.SaveImage;
var InputString:string;
begin
 InputString:= InputBox('Nom du fichier', 'ENTREZ', 'LabyPerso.bmp');
 planLaby.Picture.SaveToFile(InputString);
end;

procedure TLabyrinthe.SetNbrcases(x: integer);
begin
 if x in [2..maxTab] then
 begin
  FNbrcases:=x;
  Dim.largeur:=x;
  Dim.hauteur:=x;
  RegenerAutre;
 end
end;
//----------- Le déplacement dans le labyrinthe
procedure TLabyrinthe.MovePelerin(vers:TPoint);
begin
 Pelerin.Top:=posControl.y+vers.y;
 Pelerin.Left:=posControl.x+vers.x;
end;

procedure TLabyrinthe.ChangeCoulCell(coord:TPoint;coul:TColor);
begin
 planLaby.canvas.brush.color:=coul;
```

```
  planLaby.canvas.fillrect(rect(coord.y*taille,coord.x*taille,
  (coord.y+1)*taille,(coord.x+1)*taille));
  planLaby.Update
 end;

 procedure TLabyrinthe.MoveIHM(x, y: integer; posExpl: TPoint;
  caillou: integer);
  begin
   inherited;
   MovePelerin(Point((posExpl.y)*taille, (posExpl.x)*taille));
   case
    caillou of
    markaller:
    begin
     ChangeCoulCell(Point(x,y),CoulAller);
     Pelerin.Color:=clRed;
    end;
    markretour:
    begin
     ChangeCoulCell(Point(x,y),CoulRetour);
     Pelerin.Color:=clBlue;
    end;
    carrefour,sortie:ChangeCoulCell(Point(x,y),CoulAller);
   end;
  end;

end.
```

Aspect après plusieurs appuis sur le bouton Déplacer :



Extension de la solution à une nouvelle version sous forme de composant d'animation réutilisable à déposer dans la palette des composants Delphi dans l'onglet 'Exemples', il est construit par association sur un TPanel du Labyrinthe proprement dit dessiné sur un TImage, de 4 TSpeedButton permettant d'intervenir sur la génération d'un nouveau labyrinthe et sur l'animation de la recherche dans le labyrinthe et enfin d'unTSpinEdit qui permet de changer la taille du labyrinthe (valeur entre 2 et 40), l'animation est assurée par un TTimer :



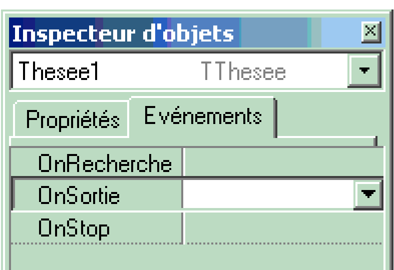
Propriétés :
ColorFond = couleur des couloirs.

ColorMurs = couleur des murs.

Propriété :
Taille = nombre de cellules du côté du carré du labyrinthe (varie de 2 jusqu'à 40).

Le composant est sensible à 3 événements :

OnRecherche : qui se produit dès que le pèlerin commence sa recherche de sortie dans le labyrinthe.
OnSortie : qui se déclenche dès que le pèlerin a trouvé une sortie dans le labyrinthe.
OnStop : qui se déclenche dès que le pèlerin s'est arrêté dans l'un des 3 cas suivants : on arrête l'animation avec le bouton stop, le pèlerin s'est rendu compte qu'il n'y avait pas de sortie dans le labyrinthe, le pèlerin a trouvé une sortie dans le labyrinthe.



Dès que l'on clique sur le bouton lancer recherche, l'animation commence, le pèlerin se déplace dans le labyrinthe.

Ce composant utilise la **unit** UCreerLaby; précédente qui contient les 3 classes de base du labyrinthe :

```
unit ULabyrinthe;
{ un composant obtenu à partir du   design pattern sur les labyrinthes }
interface

 uses UCreerLaby,Messages,Classes,extctrls,Controls, graphics,buttons,
  Spin;
  {
   code Damier
     0 --> départ
     1 --> carrefour
     2 --> libre
     3 --> aller
     4 --> retour
    10 --> mur
    20 --> sortie
  }
 type
  TThesee =class(TCustomPanel)
 private
  Ftaille:integer;
  Fstop:boolean;
  FOnStop,FOnRecherche,FOnSortie:TNotifyEvent;
  TheseeExplor,Command:Tpanel;
  Plan:Timage;
  LabyMinos:TLabyrinthe;
  Tempo:TTimer;
  SpeedBtnLancer,SpeedBtnNew,SpeedBtnClear,SpeedBtnStop:TSpeedButton;
  SpinEditCases: TSpinEdit;
  procedure Timing(Sender: TObject);
  procedure SpeedBtnLancerClick(Sender: TObject);
  procedure SpeedBtnNewClick(Sender: TObject);
  procedure SpeedBtnClearClick(Sender: TObject);
  procedure SpeedBtnStopClick(Sender: TObject);
  procedure SpinChange(Sender: TObject);
  procedure SetTaille(x:integer);
  procedure WMSizeRedim(var Msg:TWMsize); message WM_size;
  function GetColorMurs:TColor;
  function GetColorFond:TColor;
```

```
  procedure SetColorMurs(x:TColor);
  procedure SetColorFond(x:TColor);
 public
  constructor Create(AOwner: TComponent); override;
 published
  property Taille:integer read Ftaille write SetTaille;
  property ColorMurs:TColor read GetColorMurs write SetColorMurs;
  property ColorFond:TColor read GetColorFond write SetColorFond;
  property OnStop:TNotifyEvent read FOnStop write FOnStop;
  property OnSortie:TNotifyEvent read FOnSortie write FOnSortie;
  property OnRecherche:TNotifyEvent read FOnRecherche write FOnRecherche;
end;

  procedure Register;

implementation

 procedure Register;
 begin
  RegisterComponents( 'Exemples', [TThesee] );
 end;
 {---------------------- TThesee -----------------------}
 constructor TThesee.Create(AOwner: TComponent);
 var i:integer;
 begin
  inherited;
  self.Width:=410;
  self.Height:=452;
  self.BevelInner:=bvLowered;
  self.BevelOuter:=bvNone;
  self.Font.Name:='Times New Roman';
  self.Font.Size:=8;
  self.parent:=TWinControl(AOwner);
  Command:=Tpanel.Create(self);
  Command.parent:=self;
  Command.SetBounds(0,0,410,40);
  Command.Align:=alTop;
  SpeedBtnLancer:=TSpeedButton.Create(Command);
  SpeedBtnLancer.SetBounds(2,10,90,25);
  SpeedBtnLancer.Caption:='Lancer recherche';
  SpeedBtnNew:=TSpeedButton.Create(Command);
  SpeedBtnNew.SetBounds(93,10,100,25);
  SpeedBtnNew.Caption:='Nouveau Labyrinthe';
  SpeedBtnClear:=TSpeedButton.Create(Command);
  SpeedBtnClear.SetBounds(194,10,100,25);
  SpeedBtnClear.Caption:='Repartir au début';
  SpeedBtnStop:=TSpeedButton.Create(Command);
  SpeedBtnStop.SetBounds(295,10,60,25);
  SpeedBtnStop.Caption:='Stop';
  SpeedBtnStop.Enabled:=false;
  for i:=0 to Command.ComponentCount-1 do
   if Command.Components[i] is TSpeedButton then
    with (Command.Components[i] as TSpeedButton)do
    begin
     parent:=Command;
     flat:=true;
     cursor:=crHandPoint
    end;
  SpeedBtnLancer.OnClick:=SpeedBtnLancerClick;
  SpeedBtnNew.OnClick:=SpeedBtnNewClick;
```

```
 SpeedBtnClear.OnClick:=SpeedBtnClearClick;
 SpeedBtnStop.OnClick:=SpeedBtnStopClick;
 SpinEditCases:=TSpinEdit.Create(Command);
 SpinEditCases.parent:=Command;
 SpinEditCases.SetBounds(358,10,45,25);
 SpinEditCases.MaxValue:=maxTab;
 SpinEditCases.MinValue:=2;
 SpinEditCases.Value:=maxTab;
 SpinEditCases.Increment:=2;
 SpinEditCases.OnChange:= SpinChange;
 Plan:=Timage.Create(self);
 Plan.parent:=self;
 Plan.AutoSize:=false;
 Plan.SetBounds(0,42,410,410);
 Plan.canvas.brush.color:=clSilver;
 Plan.Align:=alClient;
 Plan.canvas.fillrect(rect(0,0,width,height));
 Ftaille:=maxTab;
 Fstop:=true;
 TheseeExplor:=Tpanel.Create(self);
 TheseeExplor.parent:=self;
 TheseeExplor.Color:=clRed;
 LabyMinos:=TLabyrinthe.Create(Plan);
 LabyMinos.Pelerin:=TheseeExplor;
 TheseeExplor.Width:= LabyMinos.taille;
 TheseeExplor.Height:= LabyMinos.taille;
 LabyMinos.posPelerin:=Point(0,0);
 LabyMinos.MovePelerin(Point(0,0));
 Tempo:=TTimer.Create(self);
 Tempo.Interval:=50;
 Tempo.Enabled:=false;
 Tempo.OnTimer:=Timing;
end;

procedure TThesee.Timing(Sender: TObject);
begin
 if not LabyMinos.sortieFound and not LabyMinos.Nosortie
  and not fstop then
      labyminos.deplacer
 else
 begin
  tempo.Enabled:=false;
  if assigned(FOnStop) then
      FOnStop (self);
  if LabyMinos.sortieFound then
  begin
   if assigned(FOnSortie) then
        FOnSortie (self);
   SpeedBtnStop.Enabled:=false;
   SpeedBtnLancer.Enabled:=false;
  end;
  if LabyMinos.Nosortie then
  begin
   SpeedBtnStop.Enabled:=false;
   SpeedBtnLancer.Enabled:=false;
  end
 end
end;

procedure TThesee.SpeedBtnLancerClick(Sender: TObject);
```

Gestionnaire de l'événement Ontimer de l'objet Tempo de type TTimer :

Toutes les 50 ms le pèlerin se déplace d'une cellule dans le labyrinthe.

Gestionnaire de l'événement Ontimer de l'objet Tempo de type TTimer :

Si une sortie a été trouvée, ou il n'y a pas de sortie, ou on a demandé l'arrêt de l'animation, le timer se désactive et tout est stoppé.

```
begin
 Fstop:=false;
 Tempo.Enabled:=true;
 SpeedBtnStop.Enabled:=true;
 SpeedBtnStop.Caption:='Stop';
 if assigned(FOnRecherche) then
    FOnRecherche (self)
end;

procedure TThesee.SpeedBtnNewClick(Sender: TObject);
begin
 LabyMinos.RegenerAutre;
 SpeedBtnLancer.Enabled:=true;
end;

procedure TThesee.SpeedBtnClearClick(Sender: TObject);
begin
 LabyMinos.RegenerMeme;
 SpeedBtnLancer.Enabled:=true;
end;

procedure TThesee.SpeedBtnStopClick(Sender: TObject);
begin
 if SpeedBtnStop.Caption='Stop' then
 begin
  Fstop:=true;
  SpeedBtnStop.Caption:='Continuer';
 end
 else
 begin
  Fstop:=False;
  SpeedBtnStop.Caption:='Stop';
  Tempo.Enabled:=true;
  if assigned(FOnRecherche) then
         FOnRecherche (self)
 end
end;

procedure TThesee.SpinChange(Sender: TObject);
begin
 try
  if not odd(SpinEditCases.Value) then
   if SpinEditCases.Value in [2..maxTab] then
    LabyMinos.Nbrcases:= SpinEditCases.Value
 except
  LabyMinos.Nbrcases:=2
 end
end;

procedure TThesee.SetTaille(x: integer);
begin
 if x in [2..maxTab] then
 begin
  if odd(x) then
   x:=x+1;
  LabyMinos.Nbrcases:=x;
  Ftaille:=x;
  SpinEditCases.Value:=x;
 end
end;
```

```
procedure TThesee.WMSizeRedim(var Msg: TWMsize);
begin
 self.Width:=410;
 self.Height:=452;
end;

function TThesee.GetColorFond: TColor;
begin
 result:=LabyMinos.CoulFond
end;

function TThesee.GetColorMurs: TColor;
begin
 result:=LabyMinos.CoulMur
end;

procedure TThesee.SetColorFond(x: TColor);
begin
 LabyMinos.CoulFond :=x;
 LabyMinos.RegenerMeme;
end;

procedure TThesee.SetColorMurs(x: TColor);
begin
 LabyMinos.CoulMur:=x;
 LabyMinos.RegenerMeme;
end;

end.
```