DELPHI 5

Support de formation (3/3)

Perfectionnement

1. DESSIN DE GRAPHIQUES AVEC LE CANEVAS

Les images graphiques sont crées à la conception du projet (composant de type TImage avec propriété "picture" définie ou de type "TShape") ou dessinées à l'exécution.

Le canevas permet d'éviter d'utiliser des fonctions bas-niveau de Windows. C'est à la fois une propriété et un objet (TCanvas) qui possède d'intéressantes propriétés (ou objets). Il dispose d'un crayon (**pen**) pour le dessin de lignes, d'un pinceau (**brush**) pour le remplissage et d'une police (**font**) pour les caractères.

1.1 Propriétés principales

1.1.1 Brush (pinceau)

1.1.1.1 Style

La propriété Style d'un pinceau détermine le motif du pinceau pour peindre l'arrière-plan des fenêtres ou des formes graphiques. Le tableau suivant illustre les différentes valeurs possibles de Style et le motif résultant :

HACHURE	MOTIF
bsSolid	foncé
bsClear	clair
bsBDiagonal	diagonale montante
bsFDiagonal	diagonale descendante
bsCross	grillage horizontal-vertical
bsDiagCross	grillage en diagonales
bsHorizontal	lignes horizontales
bsVertical	lignes verticales

1.1.1.2 Color

C'est la couleur de fond (voir le chapitre "Pixels)

1.1.1.3 Bitmap

On peut définir soi-même un motif de fond défini comme bitmap de 8X8 et qui va se répéter autant de fois que nécessaire.

1.1.2 Font

La hauteur est indiquée par la propriété Height, la police par la propriété Name, la taille en points par la propriété Size, la couleur par la propriété Color, et les attributs de la fonte (gras, italique, etc.) par la propriété Style.

Propriétés du style:

VALEUR	SIGNIFICATION
fsBold	gras
fsItalic	italique
fsUnderline	souligné
fsStrikeout	barré

La propriété Style est un ensemble, elle peut donc contenir plusieurs valeurs. Une fonte peut, par exemple, être en gras et en italique.

1.1.3 Pen (crayon)

La couleur du crayon est spécifiée par la propriété Color. La largeur en pixels de la ligne tracée est indiquée par la propriété Width.

1.1.3.1 Style

Le motif de la ligne (solide, pointillé, etc) est spécifié par la propriété Style.

STYLE	SIGNIFICATION
psSolid	Le crayon dessine un trait continu.
psDash	Le crayon dessine un trait composé d'une série de tirets.
psDot	Le crayon dessine un trait composé d'une série de points.
psDashDot	Le crayon dessine un trait composé d'une alternance de tirets et de points.
psDashDotDot	Le crayon dessine un trait composé d'une série de combinaisons trait-point-point.
psClear	Le crayon dessine un trait ne laissant pas de marque visible.
psInsideFrame	Le crayon dessine une ligne dans le contour d'une forme fermée spécifiée par le rectangle de délimitation.

1.1.3.2 Mode

La propriété Mode spécifie la couleur de la ligne par rapport aux pixels qu'elle recouvre. Par exemple, pour colorier la ligne en utilisant la couleur indiquée par la propriété Color, basculez Mode à pmCopy. Pour colorier la ligne en inversant la couleur recouverte, basculez Mode à pmNot.

MODE	COULEUR DU PIXEL
pmBlack	Toujours noir.
pmWhite	Toujours blanc.
pmNop	Inchangé.
pmNot	Inverse de la couleur de l'écran.
pmCopy	Couleur de crayon spécifiée par la propriété Color.
pmNotCopy	Inverse de la couleur du crayon.
pmMergePenNot	Combinaison de la couleur du crayon et de l'inverse de la couleur de l'écran.
pmMaskPenNot	Combinaison des couleurs communes au crayon et à l'inverse de l'écran.
pmMergeNotPen	Combinaison de la couleur de l'écran et de l'inverse de la couleur du crayon.
pmMaskNotPen	Combinaison des couleurs communes à l'écran et à l'inverse du crayon.
pmMerge	Combinaison de la couleur du crayon et de l'écran.
pmNotMerge	Inverse de la combinaison pmMerge de la couleur du crayon et de l'écran.
pmMask	Combinaison des couleurs communes au crayon et à l'écran.
pmNotMask	Inverse de la combinaison pmMask de la couleur du crayon et de l'écran.
pmXor	Combinaison des couleurs du crayon et de l'écran mais n'apparaissant pas dans les deux.
pmNotXor	Inverse de la combinaison pmXor des couleurs du crayon et de l'écran mais n'apparaissant pas dans les deux.

1.1.4 Pixels

On peut ici lire ou attribuer la couleur de chaque point. La couleur est de type TColor et est exprimée sur 4 octets dont 3 donnent l'intensité (de 0 à 255) pour les 3 couleurs additives: rouge, vert et bleu.

Il existe des prédéfinitions de couleurs

VALEUR	SIGNIFICATION
clBlack	Noir
clMaroon	Marron
clGreen	Vert
clOlive	Vert olive
clNavy	Bleu marine
clPurple	Violet
clTeal	Turquoise
clGray	Gris
clSilver	Argent
clRed	Rouge
clLime	Vert clair
clBlue	Bleu
clFuchsia	Fuchsia
clAqua	Eau
clWhite	Blanc
clBackground	Couleur courante de fond Windows
clActiveCaption	Couleur courante de la barre de titre de la fenêtre active
clInactiveCaption	Couleur courante de la barre de titre des fenêtres inactives
clMenu	Couleur de fond courante des menus
clWindow	Couleur de fond courante des fenêtres
clWindowFrame	Couleur de fond courante des cadres de fenêtre
clMenuText	Couleur courante du texte des menus
clWindowText	Couleur courante du texte des fenêtres
clCaptionText	Couleur courante du texte dans la barre de titre de la fenêtre active
clActiveBorder	Couleur courante de la bordure de la fenêtre active
clInactiveBorder	Couleur courante de la bordure des fenêtres inactives
clAppWorkSpace	Couleur courante de l'espace de travail de l'application
clHighlight	Couleur de fond courante du texte sélectionné
clHightlightText	Couleur courante du texte sélectionné
clBtnFace	Couleur courante d'une face de bouton
clBtnShadow	Couleur courante de l'ombre projetée par un bouton
clGrayText	Couleur courante du texte grisé
clBtnText	Couleur courante du texte d'un bouton
clInactiveCaptionText	Couleur courante du texte dans la barre de titre d'une fenêtre inactive
clBtnHighlight	Couleur courante d'un bouton en surbrillance

Les autres couleurs peuvent êtres obtenues par une fonction RGB dont voici l'entête:
function RGB(Rouge: Byte; Vert: Byte; Bleu: Byte): LongInt;

Si toutes les intensités sont à 0, la couleur est noire. On obtient le blanc avec des valeurs maximales partout (255).

1.2 Méthodes principales

1.2.1 MoveTo (déplacer)

La position courante du crayon est indiquée par la propriété PenPos. Pour déplacer le crayon, appelez la méthode MoveTo.

procedure MoveTo(X, Y: Integer);

1.2.2 TextOut (affichage de texte)

Pour afficher du texte, appelez la méthode TextOut (pour déterminer si le texte peut s'afficher à l'intérieur d'une zone définie, utilisez TextHeight et TextWidth).

procedure TextOut(X, Y: Integer; const Text: string);

1.2.3 LineTo (ligne droite)

procedure LineTo(X, Y: Integer);

1.2.4 PolyLine (suite de lignes)

procedure Polyline(Points: array of TPoint);

1.2.5 Arc ou Chord

Pour dessiner des courbes, appelez les méthodes Arc ou Chord.

La méthode Arc dessine un arc sur le canevas en suivant le périmètre de l'ellipse circonscrit au rectangle spécifié. Les coordonnées (X1, Y1 et X2, Y2) définissent le rectangle qui circonscrit l'arc. L'arc débute à l'intersection du bord de l'ellipse et de la droite passant par le centre de l'ellipse et le point initial (X3, Y3). Le tracé de l'arc se fait dans le sens inverse des aiguilles d'une montre et s'arrête lorsque le point d'intersection entre le bord de l'ellipse d'une part, et la ligne qui passe par le centre de l'ellipse et le point de fin (X4, Y4) d'autre part, est atteint.

procedure Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);

La méthode Chord trace une corde sur le canevas restant circonscrite au rectangle spécifié en connectant deux points d'une ellipse. Les coordonnées écran (X1, Y1) et (X2, Y2), exprimées en pixels, définissent le rectangle qui circonscrit la corde. (X3,Y3) est le point de départ du tracé et (X4, Y4) est le point d'arrivée.

procedure Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);

1.2.6 Rectangle

procedure Rectangle(X1, Y1, X2, Y2: Integer);

1.2.7 RoundRect (rectangle aux coins arrondis)

La méthode RoundRect dessine dans le canevas, sensiblement de la même manière que la méthode Rectangle, un rectangle dont l'angle supérieur gauche se trouve au point (X1, Y1) et l'angle inférieur droit (X2, Y2). Mais, RoundRect dessine les angles avec des quarts d'ellipse de largeur X3 et de hauteur Y3.

procedure RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer);

1.2.8 Ellipse

La méthode Ellipse dessine une ellipse définie par le rectangle qui la circonscrit sur le canevas. Les coordonnées en pixels de l'angle supérieur gauche du rectangle sont (X1, Y1), et celles de l'angle inférieur droit sont (X2, Y2). Un cercle est dessiné si les points définis aboutissent à une forme carrée.

procedure Ellipse(X1, Y1, X2, Y2: Integer);

1.2.9 Pie (secteur)

La méthode Pie dessine, dans le canevas, la section d'une ellipse délimitée par le rectangle de coordonnées (X1, Y1) et (X2, Y2). La section dessinée est déterminée par les deux lignes partant du centre et passant par les points (X3, Y3) et (X4, Y4).

procedure Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Longint);

1.2.10 Polygon (polygone)

La méthode Polygon dessine dans le canevas une série de lignes en connectant les points transmis dans Points (de la même manière que la méthode PolyLine), puis ferme la forme en traçant une ligne entre le dernier et le premier point. Après avoir dessiné toute la forme, Polygon la remplit en utilisant le pinceau en cours.

procedure Polygon(Points: array of TPoint);

1.2.11 FillRect (remplissage rectangulaire)

Pour remplir une zone rectangulaire avec le motif défini par Brush, appelez FillRect.

```
procedure FillRect(const Rect: TRect)

TRect = record
case Integer of
        0: (Left, Top, Right, Bottom: Integer);
        1: (TopLeft, BottomRight: TPoint);
end;
```

Le type TRect définit un rectangle. Les coordonnées sont spécifiées soit sous la forme de quatre entiers indiquant la position des bords gauche, supérieur, droit et inférieur du rectangle, soit sous la forme de deux points donnant la position en pixels des angles supérieur gauche et inférieur droit. L'origine des coordonnées en pixels correspond à l'angle supérieur gauche de l'écran.

Le type TPoint définit un emplacement sur l'écran exprimé en pixels, l'origine étant l'angle supérieur gauche. X spécifie la coordonnée horizontale du point et Y, sa coordonnée verticale.

```
TPoint = record
X: Integer;
Y: Integer;
end;
```

1.2.12 FloodFill (pot de peinture)

Pour remplir une zone entière jusqu'à ce qu'une frontière soit rencontrée, appelez FloodFill.

La méthode FloodFill remplit une zone de la surface de l'écran en utilisant le pinceau en cours (spécifié par la propriété Brush). La méthode FloodFill commence au point de coordonnées (X, Y) et continue dans toutes les directions jusqu'aux limites de couleur.

La manière de remplir la zone est déterminée par le paramètre FillStyle. Si FillStyle est à fsBorder, la surface est remplie jusqu'à ce qu'une bordure de la couleur spécifiée par le paramètre Color soit rencontrée. Si FillStyle est à fsSurface, la surface est remplie tant que la couleur spécifiée par le paramètre Color est rencontrée. Les remplissages fsSurface sont utiles pour remplir une zone ayant une bordure multicolore.

```
procedure FloodFill(X, Y: Integer; Color: TColor; FillStyle: TFillStyle);
```

1.2.13 Draw (dessin d'image existante)

Pour produire un graphique en sortie sur un canevas, tel qu'un bitmap ou un métafichier, appelez Draw.

La méthode Draw dessine sur le canevas le graphique spécifié par le paramètre Graphic au point de coordonnées écran (X, Y) exprimées en pixels. Le graphique peut être un bitmap, une icône ou un métafichier.

```
procedure Draw(X, Y: Integer; Graphic: TGraphic);
```

1.2.14 StretchDraw (dessin avec étirement)

Pour redimensionner le graphique selon une forme déterminée lors du tracé, appelez StretchDraw.

La méthode StretchDraw dessine le graphique spécifié par le paramètre Graphic dans le rectangle spécifié par le paramètre Rect. Cette méthode permet d'adapter un graphique à la taille du rectangle.

```
procedure StretchDraw(const Rect: TRect; Graphic: TGraphic);
```

1.2.15 CopyRect

Pour effectuer une copie d'une zone rectangulaire du canevas, utilisez CopyRect.

On peut spécifier le type de copie par la propriéte CopyMode du canevas:
property CopyMode: TCopyMode;

La propriété CopyMode détermine le traitement d'une image copiée sur le canevas à partir d'un autre canevas. La valeur par défaut cmSrcCopy de CopyMode signifie que les pixels du canevas source sont copiés sur le canevas cible en remplaçant l'image qui s'y trouve. En changeant CopyMode, il est possible de créer des effets spéciaux. Le tableau suivant donne la liste des valeurs possibles de CopyMode, avec leur description :

VALEUR	SIGNIFICATION
cmBlackness	Noircit l'image en sortie.
cmDstInvert	Inverse le bitmap de destination.
cmMergeCopy	Combine le motif et le bitmap source en utilisant l'opérateur booléen AND.
cmMergePaint	Combine et inverse le bitmap source avec le bitmap de destination en utilisant l'opérateur booléen OR.
cmNotSrcCopy	Copie le bitmap source inversé vers le bitmap de destination.
cmNotSrcErase	Inverse le résultat de la combinaison des bitmaps source et de destination en utilisant l'opérateur booléen OR.
cmPatCopy	Copie le motif dans le bitmap de destination en utilisant l'opérateur booléen XOR.
cmPatInvert	Combine le bitmap de destination avec le motif en utilisant l'opérateur booléen XOR.
cmPatPaint	Combine le bitmap source inversé avec le motif en utilisant l'opérateur booléen OR. Combine ensuite le résultat de cette opération avec le bitmap de destination en utilisant l'opérateur booléen OR.
cmSrcAnd	Combine les pixels des bitmaps source et de destination en utilisant l'opérateur booléen AND.
cmSrcCopy	Copie le bitmap source dans le bitmap de destination.
cmSrcErase	Inverse le bitmap de destination et combine le résultat avec le bitmap source en utilisant l'opérateur booléen AND.
cmSrcInvert	Combine les pixels des bitmaps source et de destination en utilisant l'opérateur booléen XOR.
cmSrcPaint	Combine les pixels des bitmaps source et de destination en utilisant l'opérateur booléen OR.
cmWhiteness	Blanchit le bitmap en sortie.

La méthode CopyRect copie dans l'objet canevas une partie d'image d'un autre canevas. La propriété Dest spécifie le rectangle cible du canevas cible. La propriété Canvas spécifie le canevas source. La propriété Source spécifie le rectangle source du canevas source.
procedure CopyRect(Dest: TRect; Canvas: TCanvas; Source: TRect);

1.3 Exercice

Créer un projet simple permettant de dessiner avec la souris. Commencer par la possibilité de tracer des lignes droites dans une boîte à peindre ("TPaintbox")

```
unit Unit1;
interface
uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    ExtCtrls;
type
    TForm1 = class(TForm)
    PaintBox1: TPaintBox;
    procedure PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure PaintBox1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
     procedure PaintBox1MouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
private
    { Déclarations privées }
public
    { Déclarations publiques }
end;

var
    Form1: TForm1;
    Dessine:boolean;
    Origine_x,Origine_y:Integer;

implementation
{$R *.DFM}
procedure TForm1.PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
    begin
    dessine:=true;
    Origine_x:=X;Origine_y:=Y
end;
```

```
procedure TForm1.PaintBox1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
        if Dessine=true then
          begin
          Paintbox1.Canvas.MoveTo(Origine_x,Origine_y);
          Paintbox1.Canvas.LineTo(X,Y);
          Origine_x:=X;Origine_y:=Y;
          end;
end;

procedure TForm1.PaintBox1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
        Dessine:=false;
end;

end.
```

2. Cliquer-glisser (Drag and drop)

Le but de cette opération peut être un déplacement ou une copie. Elle se fait généralement à l'intérieur d'une même fenêtre Delphi (possibilité inter-fenêtres). Il est nécessaire de prévoir 2 étapes pour mettre en œuvre ce processus. D'une part, initialiser le composant source, d'autre part programmer le composant de destination.

2.1 Composant source

La propriété "DragMode" doit être définie. Elle détermine le comportement d'un contrôle lors d'une opération glisser-lâcher. Les valeurs possibles sont :

VALEUR	SIGNIFICATION
dmAutomatic	Si dmAutomatic est sélectionnée, le contrôle est prêt à être glissé ; l'utilisateur n'a plus qu'à cliquer dessus pour le faire glisser.
dmManual	Si dmManual est sélectionnée, le contrôle ne peut être glissé tant que l'application n'a pas appelé la méthode BeginDrag.

Si la propriété DragMode d'un contrôle est à dmAutomatic, l'application peut désactiver la fonction de glisser-lâcher lors de l'exécution en basculant sa propriété DragMode à dmManual.

procedure BeginDrag(Immediate: Boolean);

La méthode BeginDrag débute le glisser d'un contrôle. Si la valeur du paramètre Immediate est True, le pointeur de la souris prend la valeur de la propriété DragCursor et le glisser débute immédiatement. Si Immediate est à False, le pointeur de la souris ne prend pas la valeur de la propriété DragCursor et le glisser ne débute pas avant que l'utilisateur ait déplacé le pointeur de la souris sur une faible distance (5 pixels). Cela permet au contrôle d'accepter des clics de souris sans qu'ils soient interprétés comme des opérations de glisser.
Pour débuter le glisser, l'application ne doit appeler la méthode BeginDrag que si la valeur de la propriété DragMode du contrôle est dmManual.

La méthode EndDrag interrompt le glisser d'un objet. Si le paramètre Drop est à True, l'objet glissé est lâché. Si le paramètre Drop est à False, il ne l'est pas et le glisser est annulé.

procedure EndDrag(Drop: Boolean);

2.2 Composant destination

Si un composant est susceptible d'accepter un objet, il doit en avertir le composant source au moyen de la procédure "DragOver". Le paramètre "Accept" sera positionné à "true". Il faut ensuite écrire le gestionnaire d'événement correspondant au lâcher de l'objet: "DragDrop".

Exercice: créer un projet permettant de visualiser des fichiers graphiques (.bmp, .wmf, .ico) par cliquer-glisser entre leur nom et un cadre de visualisation.

3. Glisser-empiler (drag and dock)

La **nouvelle** fonction appelée "glisser-empiler" peut permettre à l'utilisateur de déplacer les objets (de type TWinControl) à l'exécution, par exemple de créer une barre d'outils flottante. De plus, il est possible de prévoir un site d'empilement (DockSite) pour y fixer temporairement ces objets.

3.1 Objets empilables

Choisir un contrôle Windows et attribuer ses propriétés "DragKind" à "dkDock" et "DragMode" à "dmAutomatic" (sinon il faudra gérer méthode "BeginDrag").

Si l'on effectue maintenant un cliquer-glisser sur l'objet, il se met dans une fenêtre indépendante adaptée à sa taille. L'utilisateur peut déplacer cette fenêtre ou même la fermer (pour revoir l'objet, attribuer "visible" à "true" sur un événement utilisateur).

3.2 Sites d'empilement

C'est le contrôle qui servira de lieu de stockage des objets empilables. Il suffit de mettre sa propriété "DockSite" à "true".

Exercice: créer des barres d'outils flottantes que l'utilisateur pourra déplacer ou fixer sur un bord de la fenêtre.

4. Manipulations de fichiers

Dans certains cas, il est indispensable de manipuler les fichiers en leur état. La récupération d'informations au format hétérogène demande un traitement informatique complexe (ex: bases de données à champs variables issues d'un ancien mini-ordinateur).

4.1 Liste des méthodes disponibles

Procédure ou fonction	Description
Append	Ouvre un fichier texte existant en ajout.
AssignFile	Affecte le nom d'un fichier externe à une variable fichier.
BlockRead	Lit un ou plusieurs enregistrements d'un fichier sans type.
BlockWrite	Ecrit un ou plusieurs enregistrements dans un fichier sans type.
ChDir	Change le répertoire en cours.
CloseFile	Ferme un fichier ouvert.
Eof	Renvoie l'état de fin de fichier d'un fichier.
Eoln	Renvoie l'état de fin de ligne d'un fichier texte.
Erase	Efface un fichier externe.
FilePos	Renvoie la position en cours dans un fichier typé ou sans type.
FileSize	Renvoie la taille en cours d'un fichier, ne s'utilise pas pour les fichiers texte.
FindFirst	Lit le nom du premier fichier du répertoire spécifié
FindNext	Lit le nom du fichier suivant
Flush	Vide le tampon d'un fichier texte en sortie.
GetDir	Renvoie le répertoire en cours dans le lecteur spécifié.
IOResult	Renvoie une valeur entière indiquant l'état de la dernière fonction d'E/S effectuée.
MkDir	Crée un sous-répertoire.
Read	Lit une ou plusieurs valeurs d'un fichier dans une ou plusieurs variables.
Readln	Fait la même chose que Read puis passe au début de la ligne suivante du fichier texte.
Rename	Renomme un fichier externe.
Reset	Ouvre un fichier existant.
Rewrite	Crée et ouvre un nouveau fichier.
RmDir	Supprime un sous-répertoire vide.
Seek	Place la position en cours d'un fichier typé ou non sur le composant spécifié. Ne s'utilise pas avec les fichiers texte.
SeekEof	Renvoie l'état de fin de fichier d'un fichier texte.
SeekEoln	Renvoie l'état de fin de ligne d'un fichier texte.
SetTextBuf	Affecte un tampon d'E/S à un fichier texte.
Truncate	Tronque un fichier avec ou sans type à la position en cours dans le fichier.
Write	Ecrit une ou plusieurs valeurs dans un fichier.
Writeln	Fait la même chose que Write puis écrit un marqueur de fin de ligne dans le fichier texte.

4.2 Déclaration de fichier

Pour utiliser un fichier, il convient de le déclarer lors de l'achat des variables. Exemple:
var f: file of TypeVariable*;*

Si l'on ne connaît pas le type de données, on peut toujours dire qu'un fichier contient des octets et déclarer:
var f: file of byte;

Il est primordial d'associer un nom physique à la variable logique ainsi créée. Dans le code source, placer une instruction de la forme:
AssignFile(f,'C:\MonProjet\MonTexte.txt');

Maintenant que tout est clairement défini, nous pouvons ouvrir ce fichier, y lire et y écrire des informations.

4.3 Ecriture dans un fichier séquentiel

a) Ouvrir le fichier en écriture:
Rewrite (f);
si le fichier existe déjà, il est supprimé puis recréé.
b) Ecrire dans le fichier
Write (f, NomVariable);
c) Fermer le fichier:
CloseFile (f);

4.4 Lecture d'un fichier séquentiel

a) Ouvrir en lecture:
Reset (f);
b) Lire dans le fichier:
Read (f, NomVariable);
c) Fermer le fichier:
CloseFile (f);

4.5 Cas particulier des fichiers texte

Les fichiers texte comportent des séparateurs de lignes. Ces caractères correspondent au retour-chariot (code ANSI 10) et saut le ligne (code ANSI 13).

La déclaration peut se faire ainsi: *var f : textfile;*

L'écriture d'une ligne de texte utilise l'instruction "Writeln" et le lecture "Readln".

4.6 Exercice: encodage et décodage de fichiers

Ecrire un utilitaire permettant d'encoder ou de décoder tous types fichiers.

Information importante: sont autorisées depuis peu les clés de 128 bits, comme aux Etats-Unis.

Quelques méthodes de chiffrement:

- combinaisons (opérations sur octets): on peut par exemple ajouter le code ANSI d'un caractère de la clé tournante et le retrancher au décodage
- découpes (fonctions logiques avec un masque bit à bit): la fonction "AND" permet de traiter des sous-ensembles d'octets;
- décalages et rotations (déplacements des bits à droite ou à gauche): voir les instructions "SHR" et "SHL".

5. Utilisation du débogueur

Plusieurs types d'erreurs (ou bogues) peuvent se produire lors de la conception d'un projet:

- erreurs de syntaxe ou de logique (mises en évidence par le compilateur); on peut vérifier par l'option compiler - vérifier la syntaxe,
- erreurs d'exécution non décelées par le compilateur.

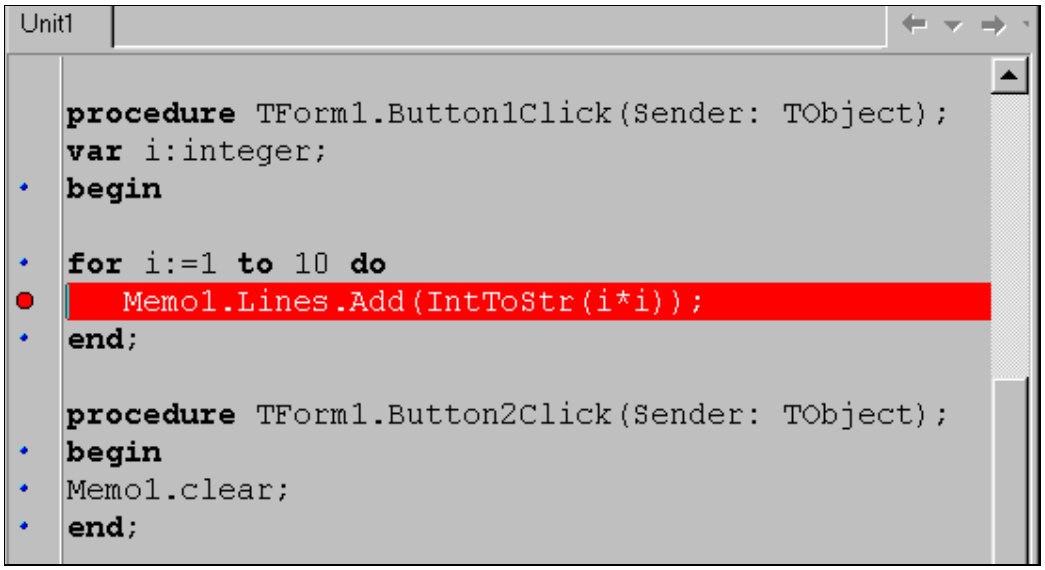
Le débogage consiste à exécuter partiellement le programme et à examiner la valeur de certaines variables au moment de l'arrêt. On peut ainsi suivre la progression et l'évolution des valeurs de celles-ci. Le processus le plus rapide est celui des points d'arrêts.

En cas de blocage, on peut réinitialiser le programme avec "Ctrl+F2".

5.1 Exécution partielle

5.1.1 Points d'arrêt

Pour poser (ou enlever) un point d'arrêt, il suffit de cliquer dans la marge de la ligne de code qui servira de point d'arrêt (ou F5).



L'exécution peut se lancer avec la touche de fonction F9. Le programme s'arrête sur l'instruction. Il est intéressant de voir simultanément la fenêtre d'exécution et l'éditeur de code.

Pour connaître la valeur d'une variable, il suffit de d'immobiliser plus de 800 millisecondes sur elle avec le pointeur de la souris: l'info-bulle affiche la valeur.

On peut poursuivre l'exécution avec F9 ou réinitialiser le programme par le menu "exécuter" ou Ctrl-F2.

5.1.2 Pas à pas

Le mode pas à pas marque des arrêts à chaque instruction de la procédure en cours. Les appels de fonctions ne font l'objet d'aucun arrêt.

Choisir Exécuter - Pas à pas ou bien F8. Pour poursuivre le pas à pas appuyer à chaque fois sur F8. Sinon, on peut lancer l'exécution du restant avec F9.

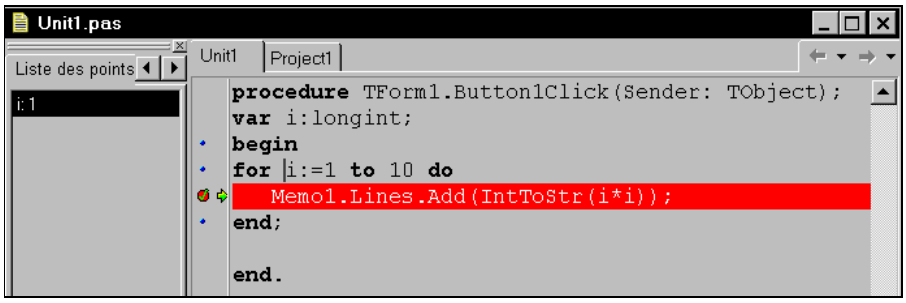
5.1.3 Pas à pas détaillé

Ce mode est identique au précédent, mais des arrêts sont effectués lors des débranchements (ex: appels de fonction). Il permet de vérifier l'exécution des fonctions.

Choisir Exécuter - Pas à pas approfondi au F7.

5.2 Points de suivi

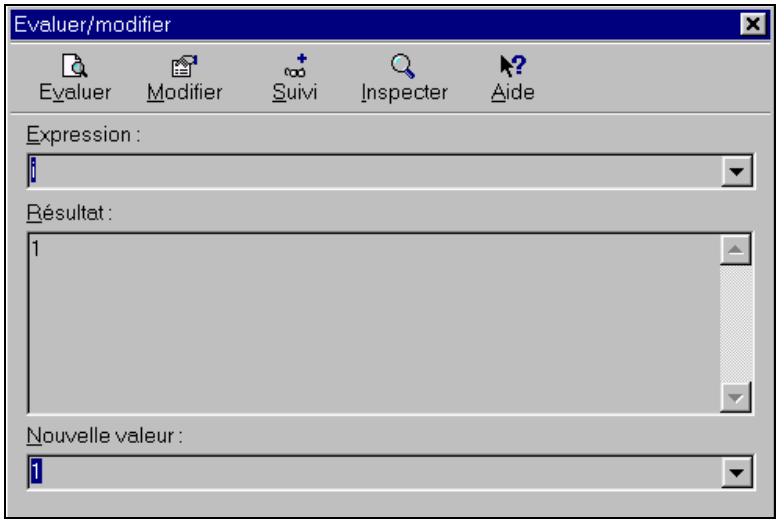
Les points de suivi sont utiles pour observer le comportement des variables. Pour la pose, sélectionner la variable et choisir "Déboguer - ajouter suivi sous curseur" dans le menu contextuel (ou Ctrl-F5).



Un double-clic sur le point de suivi permet de le modifier. La suppression peut s'effectuer par le menu contextuel.

5.3 Evaluation d'expressions

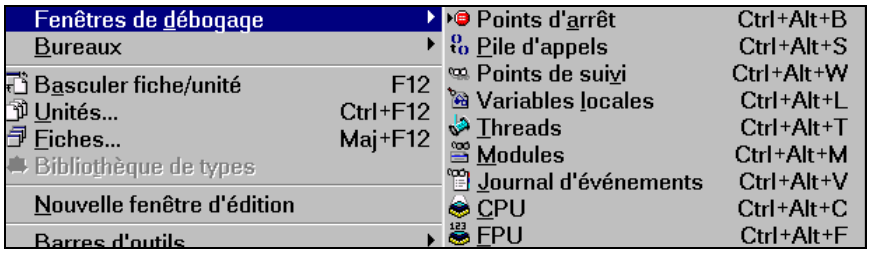
Il est également possible de demander momentanément la valeur d'une variable ou d'une expression. Choisir "Déboguer - Evaluer/modifier" dans le menu contextuel.



Le plus surprenant est ici la possibilité de changer (bouton "modifier") la valeur qui sera prise pour la poursuite de l'exécution du programme. On peut ainsi voir si le déroulement ultérieur était correct avec une autre valeur (et tester les sécurités).

5.4 Autres fonctionnalités

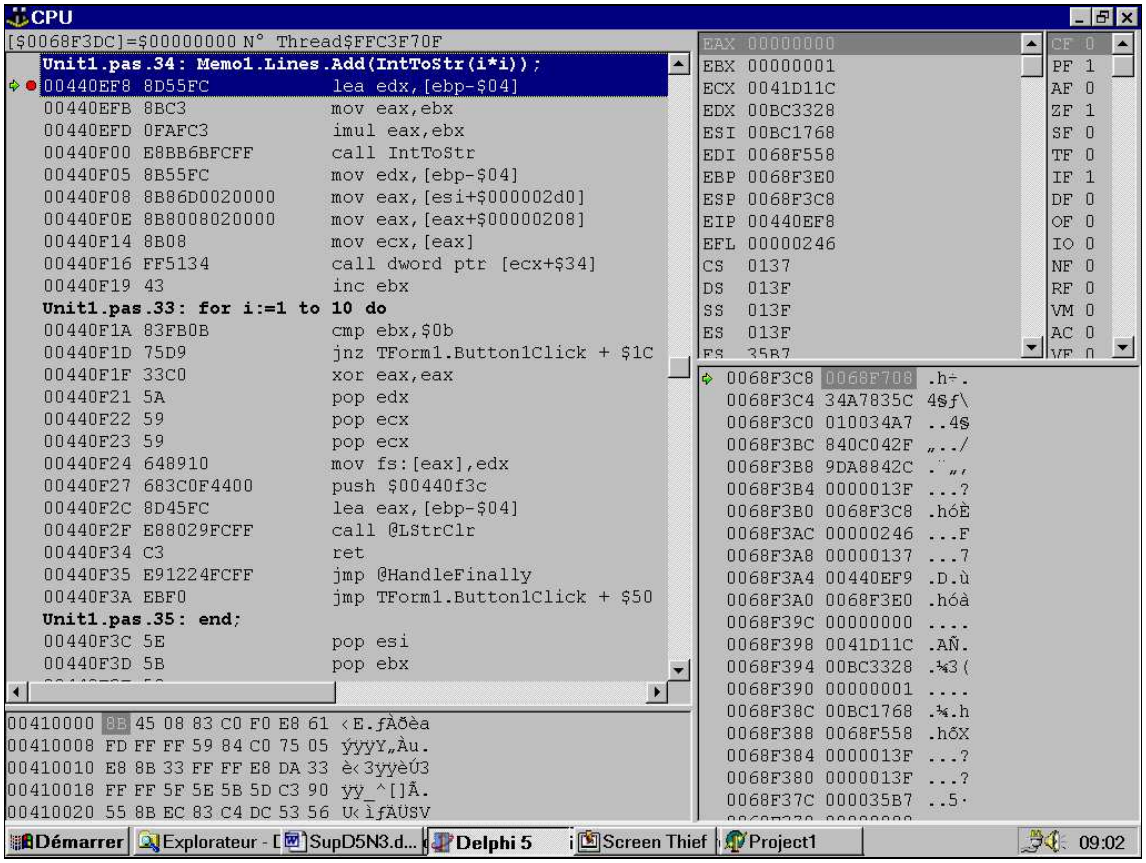
Le menu "Voir - Fenêtres de débogage" nous réserve encore des surprises:



CPU:

C'est un régal pour les "anciens" qui ont connu le langage machine et l'assembleur. On se retrouve ici au niveau du fonctionnement intime du processeur. Même les sémaphores (ou drapeaux) sont visibles.

La fenêtre CPU est composée de cinq volets séparés. Chaque volet présente un aspect de bas niveau différent de l'exécution de l'application.



Le volet de **désassemblage** (remarquable) affiche les instructions assembleur désassemblées à partir du code machine de votre application. En outre, le volet de désassemblage affiche le **code source original** du programme au dessus des instructions assembleur.

Le volet **d'affichage de la mémoire** affiche toute portion de la mémoire accessible au module exécutable en cours de chargement. Par défaut, la mémoire est présentée sous forme d'octets hexadécimaux.

Le volet de la **pile machine** affiche le contenu en cours de la pile du programme. Par défaut, la pile est affichée sous forme de nombres hexadécimaux longs (valeurs sur 32 bits).

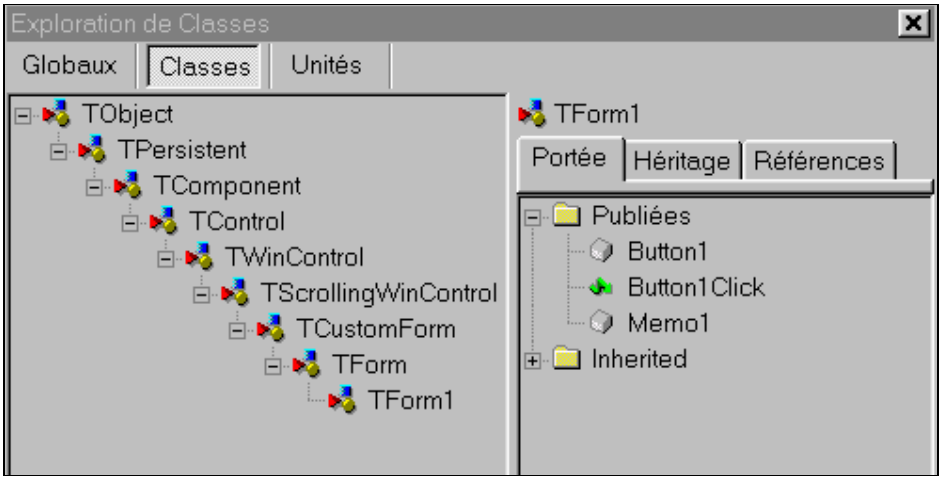
Le volet des **registres** affiche les valeurs en cours des registres du CPU.

Le volet des **indicateurs** affiche les valeurs en cours des indicateurs du CPU.

Cliquez avec le bouton droit n'importe où dans la fenêtre CPU pour accéder aux commandes spécifiques au contenu du volet en cours.

5.5 L'explorateur (anct. Scruteur)

La commande "Voir - Explorateur" permet de visualiser l'arborescence des objets de l'application et ses membres.



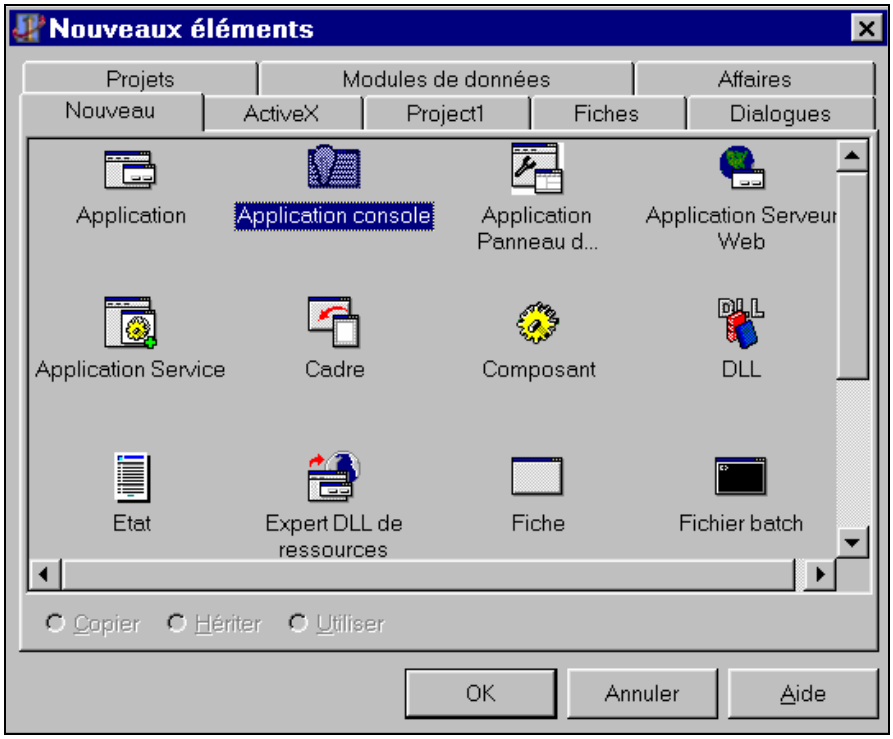
5.6 Exercice proposé

Créer un projet affichant la liste des 10 premiers carrés. Ecrire une fonction qui élève au carré. Vérifier l'utilisation du débogueur au cours de la boucle.

6. UTILISATION SOUS DOS

6.1 Création sous Windows

Choisir "Fichier – nouveau – application console"



Le listing du programme le plus simple peut être celui-ci:

```
program Project1;
{$APPTYPE CONSOLE}
uses sysutils;

begin
  // Insérer le code utilisateur ici
  writeln('hello');
  readln;
end.
```

Enregistrer puis exécuter.

7. TRAITEMENT DES EXCEPTIONS

7.1 Protection du code sensible

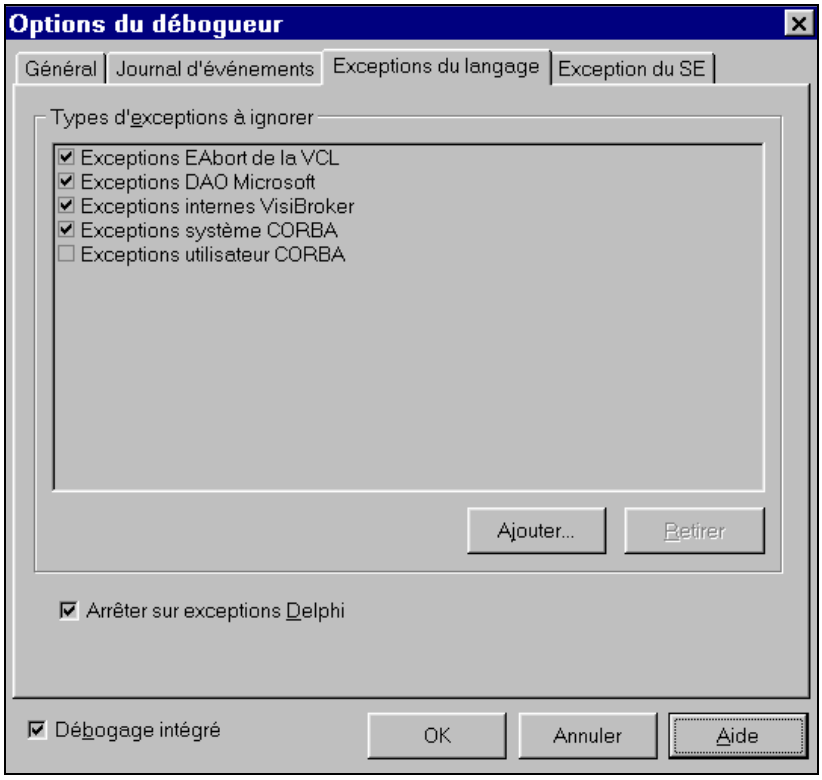
Le fait de gérer les erreurs qui pourraient survenir lors de l'exécution d'un programme permet de se prémunir contre tout "plantage" ou sortie prématurée (avec tous les risques que cela comporte: pertes de données, ...). Delphi intègre un mécanisme simple mais puissant. C'est une structure de type:

```
try
// bloc de code sensible
except
// instructions en cas d'erreur
end;
```

ou

```
try
// bloc de code sensible
finally
// instructions à exécuter avec ou sans erreur
end;
```

Dans "Outils" "Options du débogueur", on voit la boîte de dialogue suivante:



Si vous désirer gérer les erreurs, il faut décocher "Arrêter sur exceptions Delphi".

Créer un nouveau projet qui permet de multiplier ou diviser 2 nombres. Les gestionnaires d'événements peuvent s'écrire ainsi:

```
...
var
  Form1: TForm1;
  Op1,Op2,Resu:extended;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
Op1:=StrToFloat(Edit1.Text);
Op2:=StrToFloat(Edit2.Text);
Resu:=Op1*Op2;
Edit3.Text:=FloatToStr(Resu);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
try // bloc protégé
   Op1:=StrToFloat(Edit1.Text);
   Op2:=StrToFloat(Edit2.Text);
   Resu:=Op1/Op2;
   Edit3.Text:=FloatToStr(Resu);
except // gestion des erreurs

   ShowMessage('Erreur');
   close;//arrête le programme
end; //celui du try
end; //celui du begin
end. //fin de l'unité
```

Dans ce cas de figure, on ne sait pas ce qui provoque l'erreur: la mauvaise saisie de nombres ou la division par zéro. Pour le savoir, on peut tester la cause de l'erreur:

```
except // gestion des erreurs
  on EZeroDivide do
     begin
     ShowMessage('Il n''est pas possible de diviser par zéro!!!');
     close;
     end;
  else
     begin
     ShowMessage('Erreur de saisie d'un nombre');
     close;
     end;
```

7.2 Propagation d'exception

L'utilisation du mot-clé "raise" permet de propager une exception en sortant de la procédure. Il est possible de créer et d'afficher un message personnalisé.

Exemple:

```
raise Exception.Create ('Message personnalisé');
```

7.3 Gestion globale des exceptions

Il suffit de rediriger l'événement "OnException" de l'application.

Exemple:

```
Procedure TForm1.FormCreate (Sender: TObject);
begin
        Application.OnException:=AppException;
end;

Procedure TForm1.AppException(Sender:TObject; E:Exception);
begin
        Application.ShowException (E);
        Application.Terminate;
end;
```

8. Les DLL

Les DLL (Dynamic Link Libraries) sont des programmes ou données auxquels les exécutables ont accès simultanément. Il s'agit généralement de ressources graphiques (bitmap, icône ou pointeur de souris) ou de modules de programmes (bibliothèque de fonctions par exemple). Ces DLL peuvent être chargées de façon dynamique (voir l'aide en ligne) pour ne pas encombrer la mémoire vive.

8.1 Création d'une DLL

Choisir "tout fermer" dans le menu "Fichier" puis "Nouveau - DLL". Le code généré est le suivant:

```
library Project1;
uses
  SysUtils,
  Classes;
begin
end.
```

En effet, le mot clé est "library". La structure globale est semblable à un programme habituel. Il suffit de rajouter quelques éléments pour exporter des fonctions:

- la clause "export" après l'entête
- l'instruction exports suivi du nom de la fonction, d'un nom d'index optionnel ("index") et d'un surnom ou alias optionnel ("name").

Rajouter les instructions suivantes:

```
function Factorielle(n:integer):integer;export;
begin
  if n=0 then result:=1
  else
  result:=n*Factorielle(n-1);
end;

exports
  Factorielle;
```

Enregistrer dans le répertoire choisi sous le nom de "Libmath".
Choisir "Projet - Construire Libmath puis tout fermer.

8.2 Appel d'une DLL

Il est nécessaire que la DLL soit accessible: elle doit se trouver dans soit:

- le même répertoire que l'application
- le répertoire "Windows"
- le répertoire "System" de "Windows" (couramment utilisé à cet effet)
- un chemin référencé dans le "path" du DOS.

Créer un nouveau projet (dans le même dossier) comportant un "TEdit" et un bouton intitulé "Factorielle".

Déclarer l'appel de la fonction de la DLL précédente:

```
function Factorielle (i:integer):integer; external 'Libmath';
```

Créer un gestionnaire pour le clic du bouton:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Showmessage (IntToStr(Factorielle(StrToInt(Edit1.Text))));
end;
```

Enregistrer et tester. Attention à la casse!!!

8.3 Transformation d'une fiche Delphi en DLL

Il suffit d'exporter des méthodes de la fiche: création, **visualisation**, fermeture, … Beaucoup de développeurs utilisent Delphi (en secret) pour créer des petits modules sous forme de DLL utilisables avec tous autres systèmes de développement (C++, VB, Powerbuilder, etc.).

Il est donc également possible d'utiliser des DLL existantes sur le système (attention aux conventions d'appel des paramètres des procédures, voir l'aide en ligne).

9. Création de composants

L'intérêt de la création de composants est **majeure**: créer ses propres outils réutilisables. La programmation s'en trouvera vraiment simplifiée car il suffira de procéder à un assemblage d'objets. Le recensement des classes ainsi créées se fait dans la palette des composants et les propriétés des objets sont manipulables avec l'inspecteur d'objets.

La création de composants est non visuelle. Comme autrefois, il est ici nécessaire d'écrire des lignes de codes avec une aveugle confiance dans le résultat.

Types de composants:

- **Non-visuels**: Ce sont des classes (souvent descendantes de "**TComponent**") permettant de créer des objets que l'on ne verra pas à l'écran en exécution. En phase de conception, ils sont représentés par un élément de la palette des composants (ex: "TTimer").
- **Visuels**: Les classes, généralement dérivées de "**TGraphicControl**" ou "**TWinControl**" (s'il doivent prendre le focus) ont un bloc de programme qui permet de dessiner le composant à l'écran (méthode "paint").

La façon la plus simple de créer un composant est de le dériver d'un composant existant et de lui rajouter les propriétés et méthodes souhaitées.

Les composants créés par l'utilisateur se placent par défaut dans un paquet nommé "dclusr40.bpl". Les paquets sont en quelque sorte des DLL bibliothèques de DELPHI. Remarque: il est possible de créer des projets utilisant des paquets pour minimiser la taille de l'exécutable (ne pas oublier de les fournir à l'utilisateur final dans ce cas).

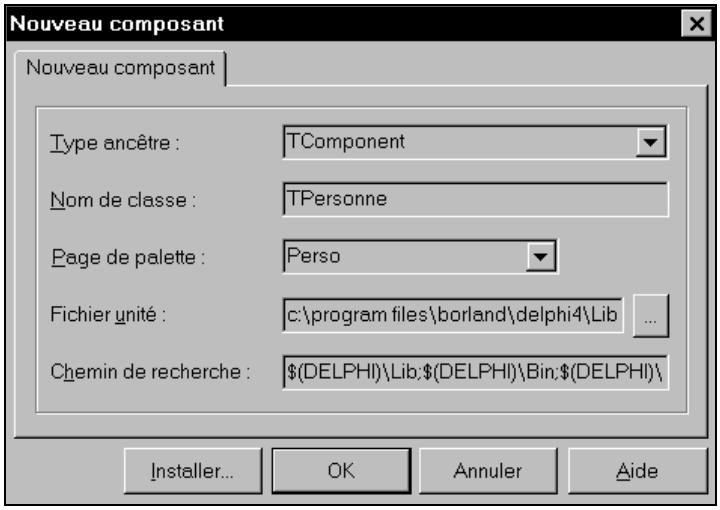
Le but de ce chapitre n'est pas de rentrer dans les détails des techniques de création (il faudrait plusieurs jours), mais de montrer un exemple simple.

9.1 Composant non-visuel

9.1.1 Création

Je veux créer une personne avec nom, prénom et âge. A chaque anniversaire, l'âge sera incrémenté.

Tout fermer et demander un nouveau composant dans le menu "fichier".



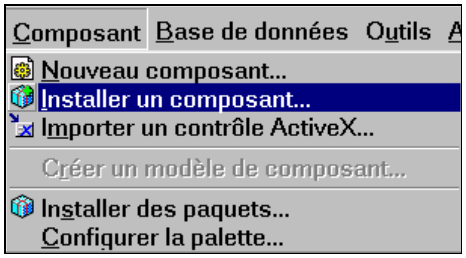
Choisit "TComponent" comme ancêtre. Le nom de la classe sera "TPersonne". Le nom de fichier unité permet de savoir où se trouve le source.

Quand je demande la création de l'unité, j'obtiens un squelette qu'il suffit de compléter:

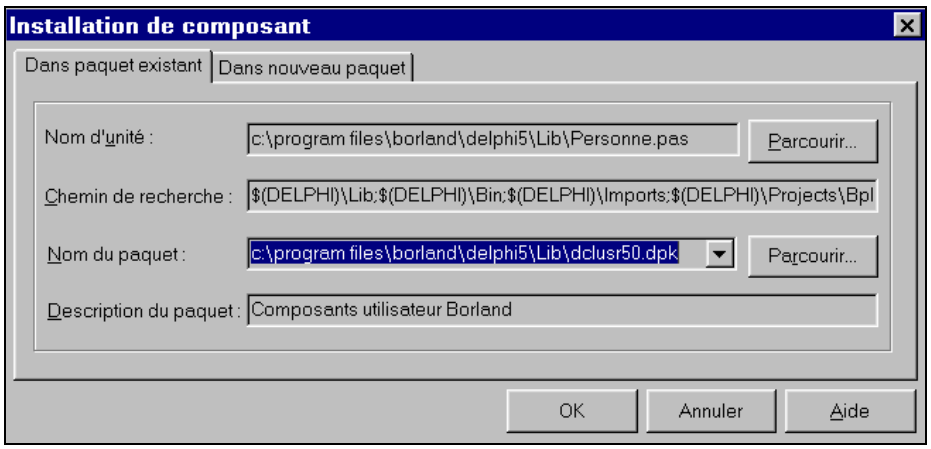
```
unit Personne;

interface

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

type
  TPersonne = class(TComponent)
  private
    { Déclarations privées }
    FNom,FPrenom:string;
    FAge:byte;
  protected
    { Déclarations protégées }
  public
    procedure Anniversaire;{ Déclarations publiques }
```

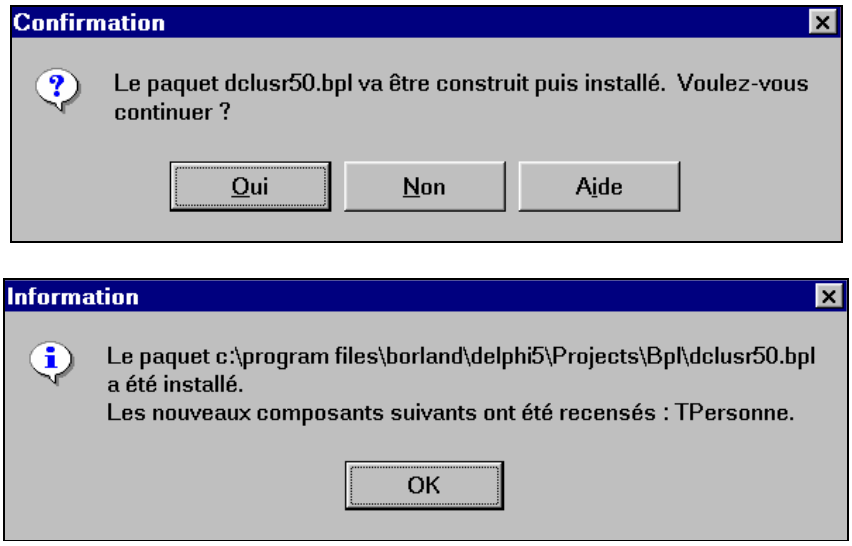
```
 published
  { Déclarations publiées }
  property Nom:string read FNom write FNom;
  property Prenom:string read FPrenom write FPrenom;
  property Age:byte read FAge write FAge;

 end;

procedure Register;

implementation
procedure TPersonne.Anniversaire;
begin
FAge:=FAge+1;
end;

procedure Register;
begin
 RegisterComponents('Perso', [TPersonne]);
end;

 end.
```

Enregistrer l'unité puis installer le composant:



Demander l'installation d'un nouveau composant dans le menu composants.

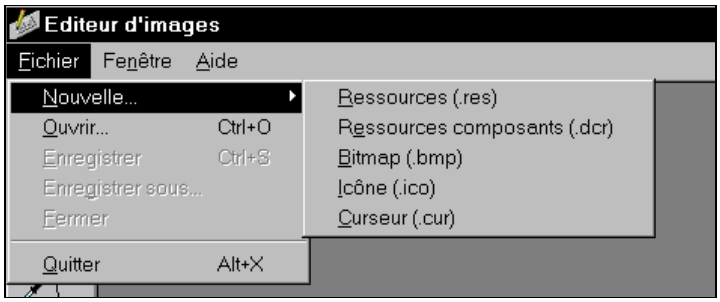




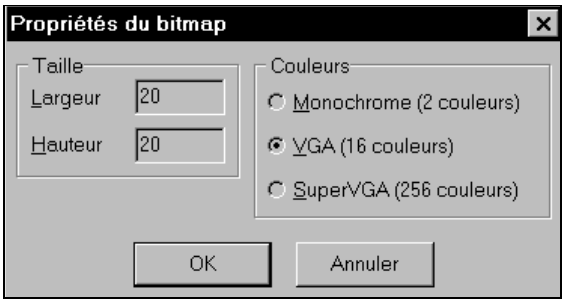
Le quitter et enregistrer. Le nouveau composant apparaît dans la palette avec une icône par défaut. Créer un projet l'utilisant.

9.1.2 Dessin de l'image du bouton représentant le composant

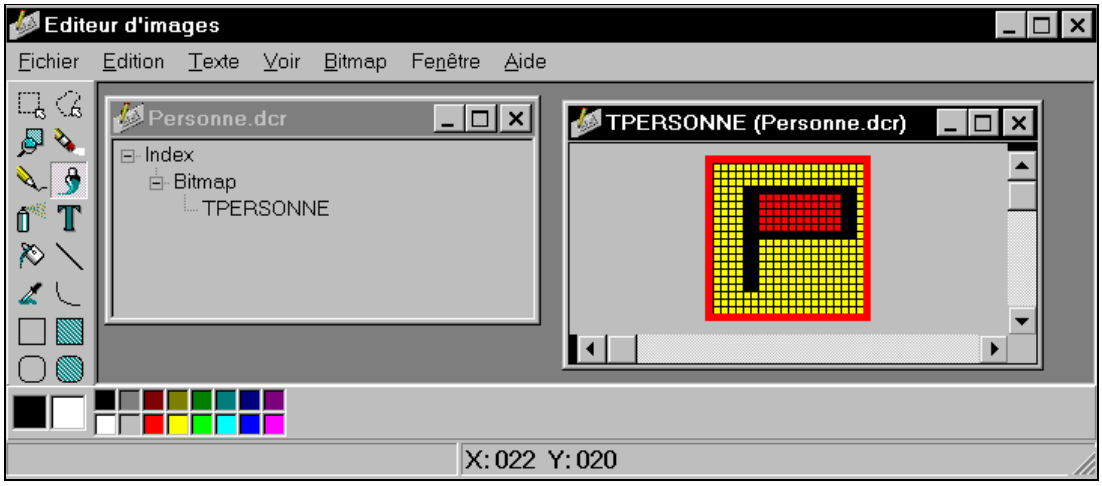
Lancer l'éditeur d'images de Delphi (Outils - Editeur d'images).



Choisir "Ressources composants (DCR)". Dans le menu contextuel de la fenêtre qui apparaît, choisir "Nouveau" "Bitmap" (20X20 pour voir les bords).



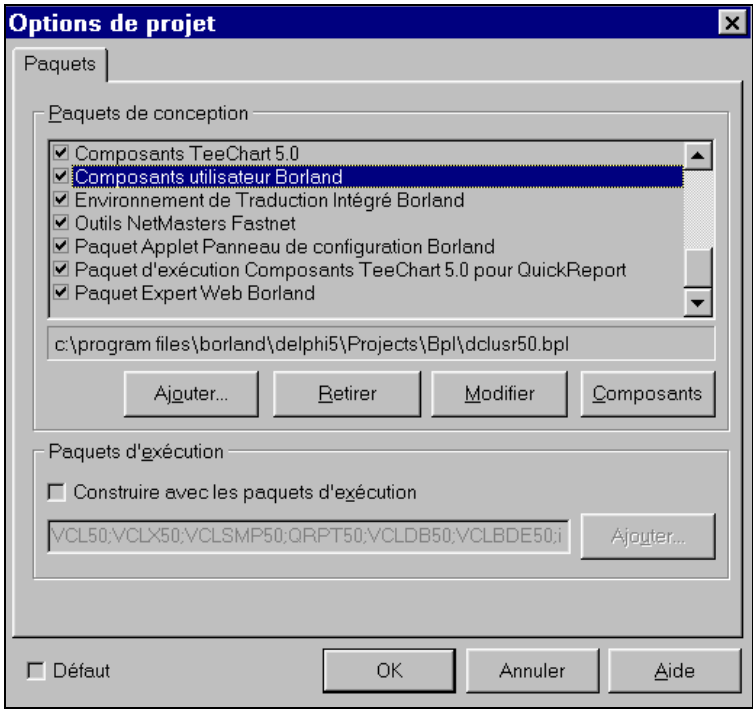
Double-cliquer sur le nom du bitmap et dessiner (agrandir):



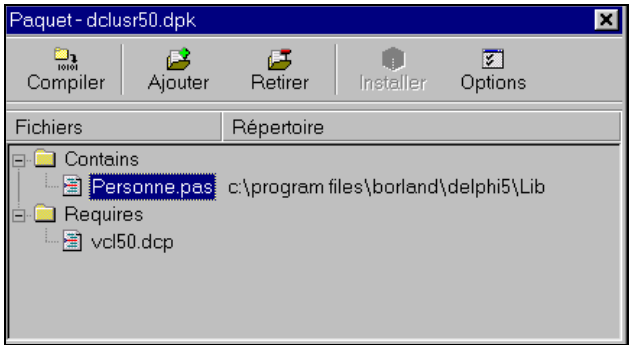
Fermer la fenêtre d'édition et renommer le bitmap avec le nom du composant (TPersonne). Enregistrer ce DCR dans le répertoire du projet avec le même nom que l'unité source de la classe du composant.

9.1.3 Recompiler le paquet

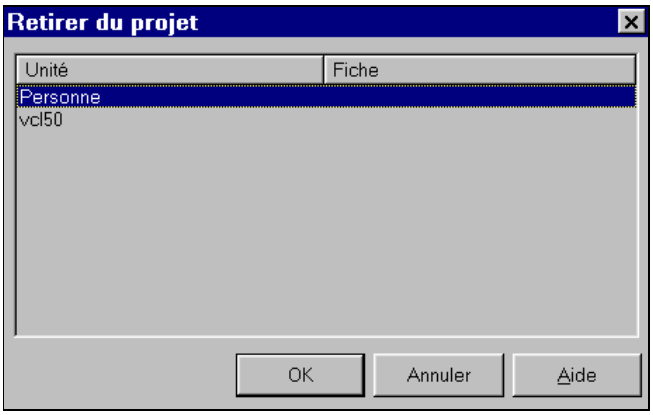
Choisir "Composants - Installer des paquets"



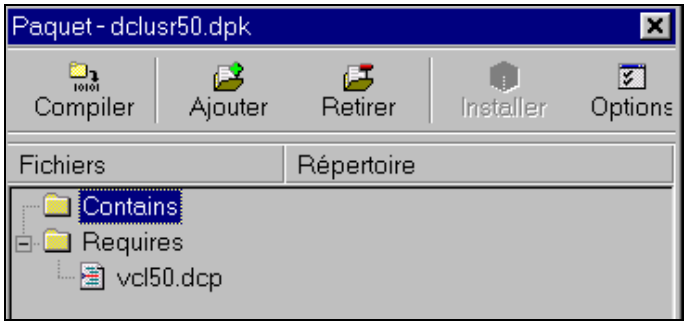
Choisir le paquet "Composants utilisateur Borland" puis cliquer sur le bouton "Modifier" puis "OK".



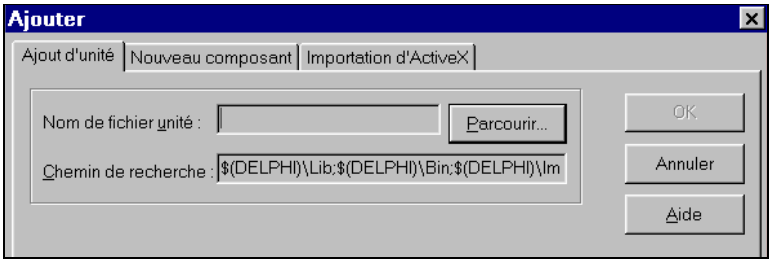
Cliquer sur "Retirer"



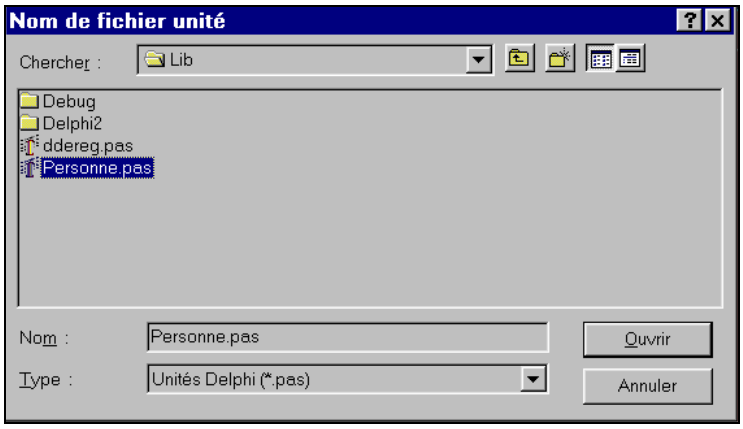
On obtient la configuration suivante:



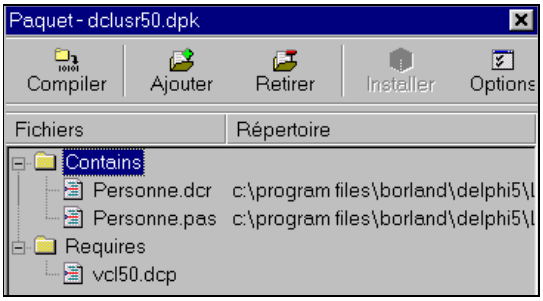
Choisir "Ajouter"



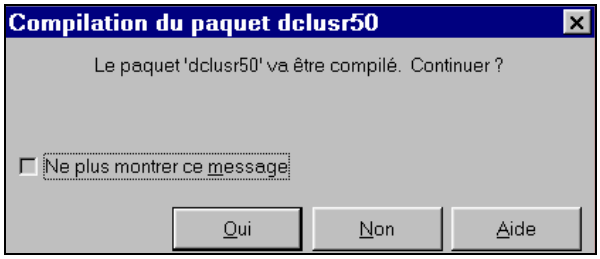
puis cliquer sur "parcourir"



Double-clic sur nom de l'unité (en .pas); on obtient l'affichage suivant:



Appuyer sur le bouton "Compiler"



Valider

Cliquer enfin sur "Oui". Vérifier l'apparition de l'image dans la palette des composants.

9.2 Composant visuel

On désire créer un composant visuel elliptique appelé "TBalle". On peut dériver "TGraphicControl" qui contient déjà des propriétés intéressantes: "Top", "Left", "Height" et "Width". Il suffit donc de surcharger le dessin ("paint").

Le listing minimal peut être le suivant:

```
unit Balle;

interface

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

type
  TBalle = class(TGraphicControl)
  private
    { Déclarations privées }
  protected
    { Déclarations protégées }

  public
    constructor Create(AOwner: TComponent);override;
    procedure paint;override;
    { Déclarations publiques }

  published
    { Déclarations publiées }
  end;

procedure Register;

implementation

  constructor TBalle.Create(AOwner: TComponent);
  begin
  inherited Create(AOwner);
  // on peut surcharger la méthode de création ici
  end;

  procedure TBalle.paint;
  begin
   Canvas.Ellipse(0,0,Width,Height);
  end;
```

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

end.
```

Options: permettre le choix de la couleur extérieure, intérieure et de l'épaisseur du contour.

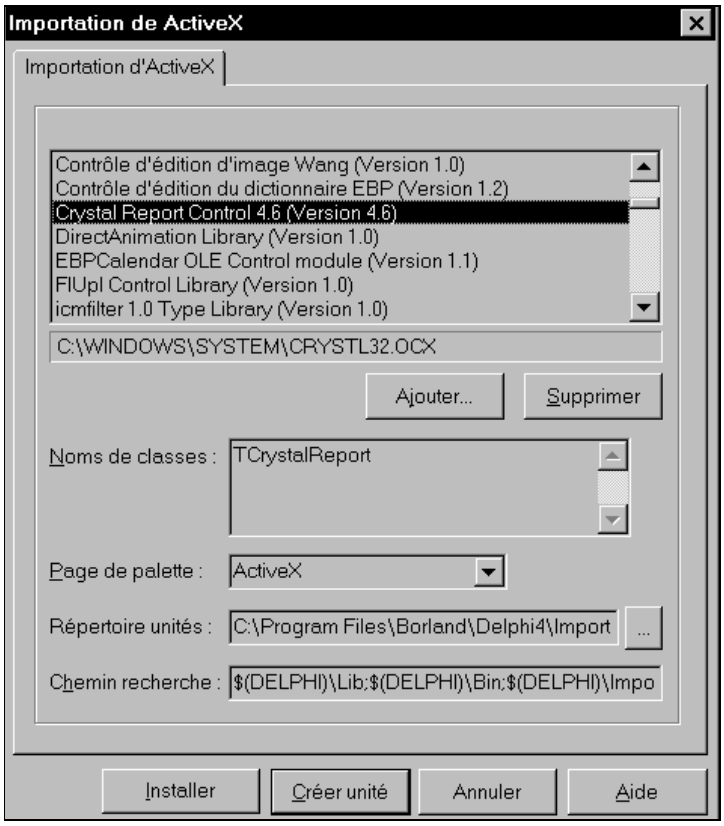
Il est facile de gérer les événements standarts en rajoutant dans la section "published" les propriétés: OnClick, OnDblClick, OnDragDrop, OnDragOver, OnDragEnd, OnMouseDown, OnMouseMove, OnMouseUp; OnEnter, OnKeyUp, OnKeyDown, OnExit, OnKeyPress,…

Exemple:

```
  published
    property OnClick;
```

9.3 Importation de contrôles Active X

Il n'y a rien de plus simple. Choisir "Composant - Importer un contrôle Active X".



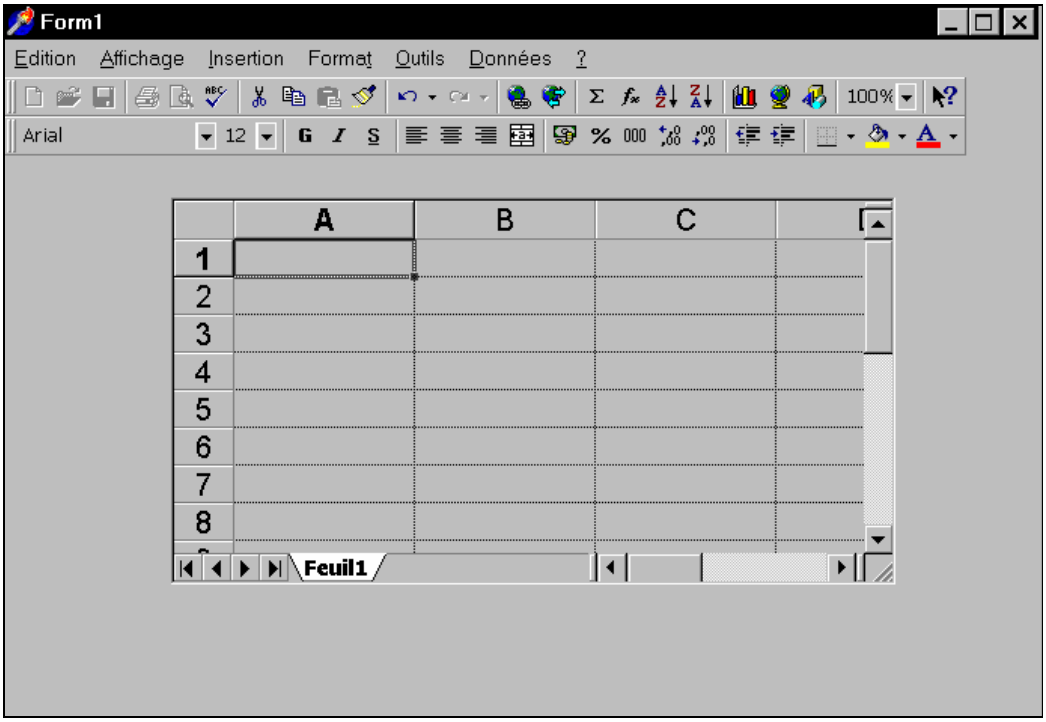
Choisir le contrôle puis cliquer sur le bouton "Installer".

10. OLE 2

10.1 Objets incorporés

Cette technique permet d'implanter sous forme d'objets des parties issues d'autres applications. C'est cas quand on met une plage Excel dans un texte Word par exemple. Ici, on pourra se servir du tableur à l'intérieur d'une fiche Delphi.

Créer un nouveau projet puis y placer un "TOleContainer". Double-cliquer et choisir "feuille Microsoft Excel" (recommencer s'il y a lieu). Une grille "excel" apparaît. C'est un objet que l'on peut ouvrir (double-clic) et modifier.



10.2 Contrôleur d'automation

On peut faire faire des tâches à un programme enregistré comme serveur d'automation: c'est le cas de Word, Excel, Access, … On se trouve dans une sorte de relation client-serveur: Delphi est le client et l'autre logiciel est serveur.

Exemple d'utilisation avec Excel:

1. créer un nouveau projet comportant un bouton
2. déclarer un variant: *var v:variant;*
3. rajouter l'utilisation de l'unité "ComObj" (indispensable!)
4. créer un objet OLE: *v:=CreateOleObject('Excel.Sheet');*
   voir dans l'aide des applications le nom des objets utilisables, c'est souvent "application" (elle s'exécutera en arrière-plan si on ne la rend pas visible)
5. rendre l'application visible: *v.Application.visible:=true;*
   on peut ici faire toute sorte de traitements par programme
6. utiliser une temporisation pour avoir le temps de voir l'objet: *sleep(10000);*
   l'utilisateur peut utiliser la feuille de calcul
7. libérer la mémoire utilisée: *v:=unassigned;*

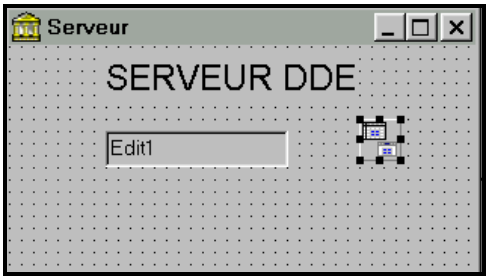
11. DDE

DDE (Dynamic Data Exchange) est le protocole pour échanger des données entre applications actives. On est ici dans une notion de type client-serveur. Une application serveur peut avoir plusieurs clients.

Nous verrons ici comment transmettre des données entre deux applications Delphi.

11.1 Le serveur DDE

Créer un nouveau projet et enregistrer (dans un dossier vide) sous "Server_p" pour le projet et "Server_f" pour la fiche. Placer un composant "TEdit", un "TDdeServerItem" et un "TLabel". Nommer la fiche "Serveur" et s'arranger pour qu'elle soit toujours au premier plan (*fsStayOnTop*).



Gérer l'événement "OnChange" de "Edit1":

```
procedure TServeur.Edit1Change(Sender: TObject);
begin
DdeServerItem1.Text:=Edit1.Text;
end;
```

Exécuter pour vérifier puis enregistrer et fermer.

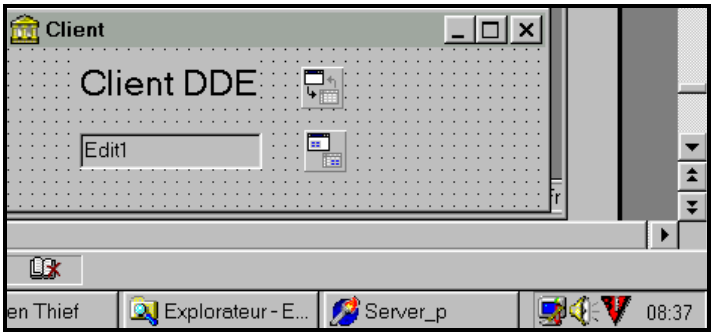
11.2 Le client DDE

Créer un projet dans le même répertoire. Enregistrer sous "Client_p" (projet) et "Client_f" (fiche).

Placer:

- un "TEdit",
- un "TDdeClientConv":
  - "ConnectMode" à "DdeManual"
  - "DDEservice" à "Server_p" (nom de l'exécutable serveur)
  - "DDETopic" à "Serveur" (nom de la fiche),
- un "TDdeClientItem":
  - "DdeConv" à "DdeConv1"
  - "DdeItem" à "DdeServerItem1" (vérifier l'acceptation).

Ensuite positionner dans "DdeClientConv1": "ConnectMode" à "DdeAutomatic". Si tout va bien, on voit l'application serveur dans la barre des tâches.



Gérer la procédure "OnChange" de " DdeClientItem1":

```
procedure TClient.DdeClientItem1Change(Sender: TObject);
begin
Edit1.Text:=DdeClientItem1.Text;
end;
```

Enregistrer tout et fermer. Exécuter l'application cliente: le serveur s'exécute automatiquement. Taper quelques caractères dans la zone d'édition de ce dernier et vérifier la réception par le client.

Remarque: On peut également se servir d'une autre application qui puisse être serveur DDE comme par exemple Excel. Un copier-coller (d'une cellule) dans "DdeService" de "DdeClient1" est facile à mettre en place).

12. Les messages Windows

En fait, Windows est basé sur la gestion des messages. Chaque événement (par exemple le déplacement de la souris dans une fenêtre) provoque l'envoi d'un message qui sera traité par Windows. Le programmeur peut intercepter ces messages et les utiliser pour effectuer des actions particulières. Il est également en mesure d'envoyer des messages par programme pour agir sur l'environnement (par exemple: fermer une fenêtre).

Un message Windows est un enregistrement de données contenant plusieurs champs exploitables. Le plus important est celui qui contient une valeur de la taille d'un entier identifiant le message. Windows définit de nombreux messages et l'unité Messages en déclare tous les identificateurs. Les autres informations utiles véhiculées par un message sont deux champs paramètre et un champ résultat.

Le premier paramètre contient une valeur 16 bits et le deuxième, une valeur 32 bits. Dans du code Windows classique, ces valeurs sont référencées par wParam et lParam, signifiant respectivement "paramètre de type word" et "paramètre de type long". Comme il arrive souvent que ces paramètres contiennent plusieurs informations, un nom particulier, tel que lParamHi, est souvent employé pour faire référence à une partie du paramètre (le mot de poids fort)

Créer un nouveau projet comprenant un mémo et un bouton ("Quitter"):

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    procedure FormActivate(Sender: TObject);
    procedure AppMessage(var Msg: TMsg; var Handled: Boolean);
    procedure Button1Click(Sender: TObject);
  private
    { Déclarations private }
  public
    { Déclarations public }
  end;

var
  Form1: TForm1;
```

```
implementation
{$R *.DFM}
procedure TForm1.AppMessage(var Msg: TMsg; var Handled: Boolean);
begin
 {if Msg.Message <>280 then
  Memo1.Lines.Add(IntToStr(Msg.Message)+' '+IntToStr(Msg.WParam)
            +' '+IntToStr(Msg.LParam) );}
  case Msg.Message of
  WM_KEYDOWN:
  Memo1.Lines.Add('Touche appuyée : '+IntToStr(Msg.WParam)+'  :');
  WM_KEYUP:
  Memo1.Lines.Add('Touche relachée : '+IntToStr(Msg.WParam)+'  :');
  WM_MOUSEMOVE:
  Memo1.Lines.Add('Souris déplacée en : '+
              IntToStr(LoWord(Msg.LParam))+' x'+
              IntToStr(HiWord(Msg.LParam))+' y');
  WM_LBUTTONDOWN:
  Memo1.Lines.Add('Bouton gauche enfoncé');
  WM_LBUTTONUP:
  Memo1.Lines.Add('Bouton gauche relaché');
  WM_RBUTTONDOWN:
  Memo1.Lines.Add('Bouton droit enfoncé');
  WM_RBUTTONUP:
  Memo1.Lines.Add('Bouton droit relaché');

  else
  {Memo1.Lines.Add('Autre message');}
  end;

end;

procedure TForm1.FormActivate(Sender: TObject);
begin
Application.OnMessage := AppMessage;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
PostMessage(Application.handle,WM_Quit,0,0);

end;

end.
```

13. TRUCS ET ASTUCES

13.1 Modèles de codes et indentation

Taper Ctrl-J dans l'éditeur de code pour choisir un modèle. On peut également taper le modèle pris Ctrl-J.

Indentation: Ctrl+Maj+I

Désindentation: Ctrl+Maj+U

13.2 Propositions d'affectation

Lors d'une affectation (:=), taper Ctrl espace.

13.3 Qui a subi un événement? (exclusivité Delphi)

13.3.1 Récupération d'une propriété de l'émetteur

Créer un projet avec 4 boutons ("TButton"). Afficher dans un composant "TEdit" la propriété "Caption" du bouton sollicité. La procédure commune pourra être:

```
procedure TForm1.affiche(Sender: TObject);
begin
Edit1.Text:=(Sender as Tbutton).Caption;
end;
```

On utilise ici le paramètre "Sender" qui correspond à l'objet qui a été cliqué. Il est nécessaire de le transtyper ("as") pour pouvoir utiliser ses propriétés.

13.3.2 Test du type d'objet émetteur

On peut également savoir quel type d'objet a été cliqué. Affecter la même procédure au composant "Edit" et modifier comme suit:

```
procedure TForm1.affiche(Sender: TObject);
begin
  if sender is Tbutton then Edit1.Text:='bouton'
  else Edit1.Text:='autre';
end;
```

L'opérateur de comparaison "is" permet de tester le type de variable.

13.3.3 Interaction sur l'émetteur

Il est possible de modifier les propriétés de l'émetteur, par exemple:

```
procedure TForm1.dit(Sender: TObject);
var UnBouton:TButton; // une variable est nécessaire
begin
UnBouton:=(Sender as TButton);
UnBouton.caption:='c''est moi';
end;
```

13.4 Affectation dynamique de procédures événementielles

Il est possible de changer, en cours d'exécution, de procédure événementielle pour un même objet. Nous allons, par exemple, faire afficher "pair" ou "impair" à un bouton, sans utiliser de compteur. Le listing peut être le suivant:

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Nouvelle(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
Showmessage('impair');
Button1.OnClick:=Nouvelle; // on réaffecte l'événement OnClick
end;

procedure TForm1.Nouvelle(Sender: TObject);
begin
Showmessage('pair');
Button1.OnClick:=Button1Click;
end;
end.
```

13.5 Test de l'application en multipostes

Si l'on ne dispose que d'un seul poste, il suffit de lancer plusieurs sessions de l'exécutable. Nous pouvons donc vérifier l'utilisation par plusieurs personnes d'une base de donnée. Chacune peut accéder au même enregistrement en lecture. La première émettant une demande de modification en reçoit l'autorisation, ce qui verrouille l'enregistrement pour les autres qui doivent attendre une validation pour pouvoir y accéder.

Exercice: vérifier que plusieurs utilisateurs de peuvent modifier le même enregistrement en même temps.

Le méthode "refresh" est ici primordiale pour actualiser l'affichage des modifications émises par les autres postes.

13.6 Laisser l'utilisateur définir lui-même ses raccourcis-clavier

Il suffit de mettre dans la boîte de dialogue un élément de type "THotkey" et d'attribuer sa propriété "Hotkey" à la propriété "ShortCut" de l'élément de menu.

13.7 Lancer une application à partir de DELPHI (en exécution)

La fonction "ExecuteFile" nous est d'un grand secours en évitant l'usage des APIs. Il est nécessaire d'utiliser l'unité FmxUtils.pas (cherchez la).

Utiliser la fonction "ExecuteFile".

13.8 Bases de données

13.8.1 Création de fiches par cliquer-glisser

- à partir de l'éditeur de champs
- à partir de l'explorateur de BDD (table entière ou champs isolés)

13.8.2 Ajout d'un champ calculé

Utiliser l'éditeur de champs puis l'événement "OnCalcFields"

TABLE DES MATIERES