**Trucs et astuces pour C++ Builder**

**INDEX :**

1. De la couleur dans une ListBox

2. Obtention des informations de version de Windows

3. Chargement d'un curseur animé depuis une ressource.

4. Texte d'un bouton sur plusieurs lignes.

5. Mettre une ProgressBar sur la StatusBar.

6. Appeler une page web avec le navigateur par défaut.

7. Afficher un texte avec un certain angle.

8. Obtenir le code des touches clavier.

9. Compiler un composant Delphi avec C++ Builder.

10. Icône représentant le composant sur la palette de composant.

11. Obtention de l'adresse IP local.

12. Saisir que des chiffres dans un TEdit.

13. Accès aux bits d'un char.

14. Séparateur décimal.

15. Modifier plusieurs composants dans une boucle.

16. Accès aux événements de l'objet Application.

17. Déplacement d'un TEdit à l'autre avec les touches flêchées.

18. Savoir si l'on est connecté à Internet.

19. Création d'une fenêtre non rectangulaire.

20. Déplacement d'une fenêtre sans barre de titre.

21. Obtenir les ressources User, System et GDI.

22. Charger un fichier wav en mémoire puis le jouer.

23. Son wave depuis une ressource.

24. ToolBar détachable.

25. Créer un raccourci.

26. Exemple d'utilisation de TPerformanceGraph.

27. Lancer l'économiseur d'écran.

28. Tapisser une Form avec un Bitmap.

29. Extraire les mots d'une AnsiString.

30. Compiler une unité Delphi.

31. Taille de la RAM physique.

32. Mettre une commande dans le menu Système.

33. Envoyer des fichiers dans la corbeille.

Copie ou déplacement de plusieurs fichiers.

34. Détection du changement de résolution de l'écran.

35. Appel de la boîte de dialogue de changement d'icônes.

36. Ouvrir votre application par l'ouverture de l'un de ses fichiers.

37. Sauvegarder la taille et la position de la fiche dans le registre.

38. Sauvegarder la taille et la position de la fiche dans un fichier ini.

39. Changer le papier peint de Windows.

Déactiver et réactiver l'écran de veille.

40. Une seule instance de l'application.

41. Comment gérer les événements d'un composant crée dynamiquement.

42. Gestion des événements OnMouseEnter et OnMouseExit.

43. Hint sur plusieurs lignes.

44. Changer la couleur d'une ProgressBar.

45. Drag and Drop de l'explorateur Windows vers votre application.

46. Réduire toutes les fenêtres du bureau.

47. Changer les couleurs System.

48. Griser le bouton fermeture et interdire le <Alt F4>.

49. Cacher l'application de la barre des tâches.

50. Lancement de votre application au démarrage de l'ordinateur.

51. Désactiver les touches Windows (Win 95).

52. Mettre une copie d'écran dans le presse papier.

53. Liste des fontes du système dans une Combobox.

54. Un bouton dans la barre des tâches pour les fiches secondaires.

55. Obtenir le numéro de série d'un dique.

56. Liste des disques dans une TStrings.

57. Obtenir la taille d'un fichier.

58. Mettre une image dans les ressources d'une application.

59. Mettre l'économiseur d'écran dans un TPanel.

60. Cacher la fenêtre principal au lancement de l'application.

61. Charger un bitmap depuis une dll.

62. Ajouter une console à votre programme

**De la couleur dans une ListBox :**

Exemple: Sur une Form (Form1) vous posez une ListBox (ListBox1) avec sa propriété Style à : lbOwnerDrawFixed. Vous y rajoutez quelques Items.

Puis le code suivant sur l'événement OnDrawItem de la ListBox :

void \_\_fastcall TForm1::ListBox1DrawItem(TWinControl \*Control,

int Index, TRect &Rect, TOwnerDrawState State)

{

TCanvas \*pCanvas = ((TListBox \*)Control)->Canvas;

if ((Index%2)==0) pCanvas->Brush->Color = clYellow;

// met 1 ligne sur 2 en jaune

if(State.Contains(odSelected)) //ligne selectionnée

{

pCanvas->Brush->Color = clRed;

pCanvas->FillRect(Rect);

pCanvas->Font->Color = clYellow;

}

else pCanvas->FillRect(Rect); //Les autres lignes

//écriture les Items.

pCanvas->TextOut(Rect.Left+5,Rect.Top+2,

((TListBox \*)Control)->Items->Strings[Index]);

}

Avec l'utilisation de ((TListBox \*)Control) au lieu de ListBox1 cette méthode est valable pour n'importe quel ListBox.

**Obtention des informations de version de Windows :**

On peut obtenir ces informations à l'aide des variables :

Win32MajorVersion, Win32MinorVersion, Win32BuildNumber, Win32Platform, Win32CSDVersion.

Exemple sur une Form (Form1) vous posez un Memo (Memo1) et un bouton (Button1) et sur l'événement OnClick de Button1 :

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

Memo1->Lines->Add("Version : " + String(Win32MajorVersion) +

"." + String(Win32MinorVersion));

Memo1->Lines->Add("Construction : " + String(LOWORD(Win32BuildNumber)));

Memo1->Lines->Add("Plateforme : " + String(Win32Platform));

Memo1->Lines->Add("Info suppl. : " + String(Win32CSDVersion));

}

**Plateforme :**

0 -> Win32s on Windows 3.1.

1 -> Win32 on Windows 95.

2 -> Win32 on Windows NT.

**Même chose mais avec l'API Windows :**

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

OSVERSIONINFO OsVer;

OsVer.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);

GetVersionEx(&OsVer);

Memo1->Lines->Add("Version : " + String(OsVer.dwMajorVersion) +

"." + String(OsVer.dwMinorVersion));

Memo1->Lines->Add("Construction : " + String(LOWORD(OsVer.dwBuildNumber)));

Memo1->Lines->Add("Plateforme : " + String(OsVer.dwPlatformId));

Memo1->Lines->Add("Info suppl. : " + String(OsVer.szCSDVersion));

}

**Chargement d'un curseur animé depuis une ressource :**

A ce jour, je n'ai trouvé qu'une solution pour charger un curseur animé à partir d'une ressource. C'est de sauvegarder la ressource dans un fichier temporaire puis de charger le curseur animé depuis le fichier temporaire.

**Exemple :**

Créer un fichier ressource "fichier.rc" à ajouter au projet:

MCursor RCDATA cursor.ani

Dans "Unit.cpp" :

const TCursor crMyCursor = 401;

//.............

//Sur l'événement OnCreate de la Form :

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

TResourceStream \*MRes = new TResourceStream((int)HInstance,"MCursor",RT\_RCDATA);

char TempFileName[MAX\_PATH];

char TempDir[MAX\_PATH];

GetTempPath(MAX\_PATH, TempDir);

GetTempFileName(TempDir, "~", 0, TempFileName);

//Création du nom de fichier temporaire

MRes->SaveToFile(TempFileName);

Screen->Cursors[crMyCursor] = LoadCursorFromFile(TempFileName);

Cursor = crMyCursor;

DeleteFile(TempFileName);

delete MRes;

}

**Une solution plus simple, mais qui ne fonctionne que sous Windows XP, 2000 et NT.**

(Je n'ai pas testé sous 2000 et NT)

Créer un fichier ressource à ajouter au projet : "Ress.rc"

#include "ress.h"

MCursor ANICURSOR cursor.ani

Un fichier entête ressource : "Ress.h"

#define ANICURSOR 21

#define MCursor 210

Fichier principal : "Unit1.cpp"

#include "ress.h"

//..........

const TCursor crAniCur = 401;

//..........

//Sur l'événement OnCreate de la Form :

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

Screen->Cursors[crAniCur]=LoadCursor(HInstance,MAKEINTRESOURCE(MCursor));

Cursor=crAniCur;

}

**Texte d'un bouton sur plusieurs lignes:**

Sur l'événement OnCreate de la Form (Form1) pour le Bouton (Button1) :

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

SetWindowLong(Button1->Handle,GWL\_STYLE,

GetWindowLong(Button1->Handle,GWL\_STYLE) | BS\_MULTILINE);

}

Le texte s'affiche sur plusieurs lignes automatiquement à l'exécution.

On peut forcer le passage à la ligne avec le caractère '\n', mais il faut affecter le texte du bouton à l'exécution.

Button1->Caption = "Ligne1\nLigne2";

**Mettre une ProgressBar sur la StatusBar :**

Sur une Form (Form1) vous posez une StatusBar (StatusBar1) et une ProgressBar (ProgressBar1)

Sur l' événement OnCreate de Form1 :

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

StatusBar1->ControlStyle = StatusBar1->ControlStyle << csAcceptsControls ;

ProgressBar1->Left = 0;

ProgressBar1->Top = 2;

ProgressBar1->Parent = StatusBar1;

}

**Appeler une page web avec le navigateur par défaut :**

#include <ShellApi.h>

//................

ShellExecute ( Handle,"open","http://www.developpez.com",NULL,NULL,SW\_NORMAL);

Le même principe peut être utilisé pour envoyer un mail:

#include <ShellApi.h>

//................

ShellExecute(Handle,"open","mailto:toto@developpez.com",NULL,NULL,SW\_NORMAL);

Ou tout simplement pour ouvrir un fichier associé à une application (ici fichier texte):

#include <ShellApi.h>

//................

ShellExecute(Handle,"open","C:\\Dossier\\Fichier.txt",NULL,NULL,SW\_NORMAL);

(N'importe quel document peut être ouvert de cette façon s'il est associé à une application.)

**Afficher un texte avec un certain angle :**

Le code ci-dessous vous permet d'afficher un AnsiString avec un certain angle. cela ne fonctionne qu'avec des polices True Type.

Canvas->Font->Size = 20;

Canvas->Font->Name = "Arial";

LOGFONT lgfont;

GetObject(Canvas->Font->Handle, sizeof(LOGFONT), &lgfont);

lgfont.lfEscapement = 450; //Angle \*10

lgfont.lfOrientation = 450; //Angle \*10

lgfont.lfOutPrecision = OUT\_TT\_ONLY\_PRECIS;

Canvas->Font->Handle = CreateFontIndirect(&lgfont);

SetGraphicsMode(Canvas->Handle, GM\_ADVANCED);

Canvas->Brush->Style = bsClear; //transparent

AnsiString text = "C++ Builder";

Canvas->TextOut(50,210, text);

**Obtenir le code des touches clavier :**

Créer un nouveau projet avec une Form (Form1) avec sa propriété KeyPreview à true et un Label (Label1)

Sur l'événement OnKeyDown de Form1 :

void \_\_fastcall TForm1::FormKeyDown(TObject \*Sender, WORD &Key,

TShiftState Shift)

{

char ch[3];

wsprintf(ch,"%X",Key);

Label1->Caption = "0x" + String(ch);

}

A l'appuie d'une touche son code s'affiche sur Label1.

**Compiler un composant Delphi avec C++ Builder :**

\* Dans le dossier de sauvegarde mettre les fichiers sources du composant. "fichier.pas" (source), "fichier.dcr" (icone qui le représentera sur la palette). parfois "fichier.res" ...

\* Sur le menu faire : "Composant" puis "Installer un composant".

\* Sur la boîte de dialogue choisir l'onglet "Dans paquet existant" ou "Dans nouveau paquet".

\* Puis faire "parcourir" sur "Nom de fichier unité" choisir "Type" : Unité Pascal (\*.PAS) et sélectionner votre fichier (fichier.pas) et faire "Ouvrir".

\* Le fichier d'icône "fichier.dcr" est automatiquement chargé.

\* Sur le champ "Nom du paquet" saisir le nom du nouveau paquet ou sélectionner un paquet existant. Puis faire Ok.

\* Faire "Ok" sur les boîtes de messages construction et compilation du paquet.

\* Faire "Installer" sur la fenêtre du paquet si ça ne s'est pas fait automatiquement.

\* Sauvegarder le paquet: Menu "Enregistrer".

(Dans le même paquet on peut mettre plusieurs composant.)

**Icône représentant le composant sur la palette de composants :**

Cette icône est mise dans un fichier ressource spécial qui a comme extension ".DCR" ce fichier doit être dans le même dossier que les fichiers sources du composant avant compilation.

**Particularités de ce fichier:**

\* Le nom du fichier doit être le même que le nom du fichier source du composant. (si "Compo.cpp" alors "Compo.dcr")

\* L'icône doit être dans un Bitmap de taille 24 x 24

\* Le nom de ce Bitmap doit être le même que celui de la classe du composant en majuscules (si "TCompo" alors "TCOMPO")

**Obtention de l'adresse IP local :**

Exemple: Sur une form (Form1) mettre un Bouton (Button1) et 2 label (Label1 et Label2) Dans l'évènement OnClick de Button1 :

#include <winsock2.h>

//.................

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

WORD wVersionRequested;

WSADATA wsaData;

char name[255];

AnsiString ip;

PHOSTENT hostinfo;

wVersionRequested = MAKEWORD( 2, 0 );

if ( WSAStartup( wVersionRequested, &wsaData ) == 0 )

{

if( gethostname ( name, sizeof(name)) == 0)

{

Label1->Caption = name; //nom de l'ordinateur

if((hostinfo = gethostbyname(name)) != NULL)

{

ip = inet\_ntoa (\*(in\_addr \*)\*hostinfo->h\_addr\_list);

Label2->Caption = ip; //adresse IP

}

}

WSACleanup( );

}

}

**Saisir que des chiffres dans un TEdit :**

Sur l'événement OnCreate de la Form (Form1) pour Edit1 :

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

SetWindowLong(Edit1->Handle,GWL\_STYLE,

GetWindowLong(Edit1->Handle,GWL\_STYLE)

| ES\_NUMBER );

}

2eme Solution: Saisir que des chiffres et au plus une virgule dans un TEdit

Sur l'événement OnKeyPress du TEdit (Edit1):

void \_\_fastcall TForm1::Edit1KeyPress(TObject \*Sender, char &Key)

{

if ((Key < '0' || Key >'9') && Key != 8 && (Key != DecimalSeparator ||

((TEdit \*)Sender)->Text.Pos(DecimalSeparator)!=0))

Key = NULL;

}

En utilisant "((TEdit \*)Sender)" au lieu de "Edit1" dans la méthode. Elle devient valable pour n'importe quel TEdit.

**Accès aux bits d'un char :**

Voici une solution pour accéder simplement aux différents bits d'un char. Pour cela nous utiliserons une union entre une structure de bits et un char.

**Exemple:**

union TSChar

{

char chr;

struct

{

unsigned char Bit0 : 1;

unsigned char Bit1 : 1;

unsigned char Bit2 : 1;

unsigned char Bit3 : 1;

unsigned char Bit4 : 1;

unsigned char Bit5 : 1;

unsigned char Bit6 : 1;

unsigned char Bit7 : 1;

};

};

//................

TSChar SChar;

SChar.chr='a';

//affectation du caractère 'a' dans l'union

Label1->Caption = SChar.Bit3;

// affichage du 4eme bit dans un label

SChar.Bit5 = 0;

//mise à 0 du 6ème bit (Cela met SChar.chr en majuscule 'A' )

**Séparateur décimal :**

Le séparateur décimal est défini dans les paramètres régionaux de Windows. ("Panneau de configuration", "Paramètre régionaux")

On peut connaître le séparateur décimal avec la variable DecimalSeparator exemple pour l'afficher dans un label : Label1->Caption = DecimalSeparator;

On peut aussi le changer en lui affectant une autre valeur ex: DecimalSeparator = ',';

Voir aide de C++ Builder : "variables de format"

**Modifier plusieurs composants dans une boucle :**

Exemple sur une Form (Form1) poser un bouton (Button1) et plusieurs TEdit. Nous allons affecter la valeur "Ok" à la propriété Text de tous les TEdits se trouvant sur la Form.

Sur l'événement OnClick de Button1 :

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

for ( int x=0 ; x<ControlCount ; x++)

if ((AnsiString)Controls[x]->ClassName() == "TEdit" )

((TEdit\*)Controls[x])->Text = "Ok";

}

**Accès aux événements de l'objet Application :**

Exemple : Sur une fiche (Form1) déposer un TShape (Shape1). Il prendra la couleur de la barre de titre en fonction que l'application est active ou pas.

Dans l'Unit1.h déclarer les 2 méthodes AppDeactivate et AppActivate:

private: // Déclarations de l'utilisateur

void \_\_fastcall AppDeactivate(TObject \*Sender);

void \_\_fastcall AppActivate(TObject \*Sender);

Dans Unit1.cpp sur l'évenement OnCreate de Form1 affecter les 2 méthodes AppDeactivate, AppActivate aux évenements correspondants. Puis définitir les 2 méthodes.

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

Application->OnDeactivate = AppDeactivate;

Application->OnActivate = AppActivate;

}

void \_\_fastcall TForm1::AppDeactivate(TObject \*Sender)

{

Shape1->Brush->Color = clInactiveCaption;

}

void \_\_fastcall TForm1::AppActivate(TObject \*Sender)

{

Shape1->Brush->Color = clActiveCaption;

}

Le principe est le même pour tous les autres événements de l'objet Application.

Sous BCB6 il existe un composant donnant accès aux événements de Application : ApplicationEvents (Onglet Supplément).

**Déplacement d'un TEdit à l'autre avec les touches flêchées :**

Nous utiliserons une méthode commune à tous les TEdits qui sera exécutée lors de l'événement OnKeyDown de chaque TEdit.

void \_\_fastcall TForm1::EditKeyDown(TObject \*Sender, WORD &Key,

TShiftState Shift)

{

TWinControl \*x;

x = FindNextControl((TWinControl \*)Sender, true, false, false);

if(Key == VK\_RIGHT) x->SetFocus();

x = FindNextControl((TWinControl \*)Sender, false, false, false);

if(Key == VK\_LEFT) x->SetFocus();

}

**Savoir si l'on est connecté à Internet :**

Pour cela nous allons utiliser la fonction "InternetGetConnectedState" de la dll "Wininet.dll". Cette façon de faire ne fonctionne pas avec tous les types de connections. A tester avec différentes configurations.

Exemple sur une Form (Form1) poser un Bouton (Button1) et un Label (Label1) :

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

typedef BOOL (WINAPI \*PF\_INETGETCONNECTEDSTATE)(LPDWORD, DWORD);

HANDLE hWinInet;

PF\_INETGETCONNECTEDSTATE pfInternetGetConnectedState;

//.............

hWinInet = LoadLibrary("WININET.DLL"); //Chargement de la dll

if(hWinInet == NULL)

{

Label1->Caption = "Impossible de charger Wininet.dll";

return;

}

pfInternetGetConnectedState = (PF\_INETGETCONNECTEDSTATE) GetProcAddress(

hWinInet, "InternetGetConnectedState");

// affectation du pointeur sur la fonction

if(pfInternetGetConnectedState == NULL)

{

Label1->Caption = "Erreur appel fonction InternetGetConnectedState";

if(hWinInet) FreeLibrary(hWinInet);

return;

}

//.............

DWORD TypeCon ;

if (pfInternetGetConnectedState(&TypeCon, 0)) //appel de la fonction

Label1->Caption = "Connecté";

else Label1->Caption = "Déconnecté";

//.............

if(hWinInet) FreeLibrary(hWinInet); //libération de la dll

}

**Création d'une fenêtre non rectangulaire :**

Pour cela nous allons créer un nouveau projet avec une Form (Form1) avec la propriété BorderStyle à bsDialog.

Sur les événements OnCreate et OnPaint de Form1

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

HRGN Region;

HRGN SubRgn;

Region = CreateRectRgn(0,0,Width,Height);

SubRgn = CreateRectRgn(Width/2-100,Height/2-100,

Width/2+300,Height/2+300);

CombineRgn(Region,Region,SubRgn,RGN\_DIFF);

SetWindowRgn(Handle,Region,true);

DeleteObject(SubRgn);

}

void \_\_fastcall TForm1::FormPaint(TObject \*Sender)

{

int bordx, bordy;

HDC dc;

RECT Rectangl;

bordx = GetSystemMetrics(SM\_CXEDGE);

bordy = GetSystemMetrics(SM\_CYEDGE);

dc = GetWindowDC(Handle);

Rectangl.left = Width/2-100-bordx;

Rectangl.top = Height/2-100-bordy;

Rectangl.right = Width/2+300+bordx;

Rectangl.bottom = Height/2+300+bordy;

DrawEdge(dc, &Rectangl, EDGE\_SUNKEN, BF\_RECT);

ReleaseDC(Handle,dc);

}

**Explication :**

\* Vous créez une région avec la fonction CreateRectRgn qui représente la surface de la Form:

Region = CreateRectRgn(0,0,Width,Height);

\* Vous créez une seconde région que vous allez combiner avec la première, ici une ronde:

SubRegion = CreateEllipticRgn(24, 11, 281, 268);

\* Puis vous les combinez ensembles avec la fonction CombineRgn

CombineRgn(Region,Region,SubRegion,RGN\_DIFF);

prototype: CombineRgn(HRGN destination, HRGN source1, HRGN source2, int Mode);

Mode et le mode de combinaison valeurs possible : RGN\_AND, RGN\_COPY, RGN\_DIFF, RGN\_OR, RGN\_XOR

\* puis vous affectez la région à la Form avec : SetWindowRgn(Handle,Region,true);

\* Vous supprimez la 2eme région qui ne sert plus. DeleteObject(SubRegion);

\* Le mieux est de faire quelques essais avec différentes dimensions et modes de combinaisons des 2 régions (vous pouvez même combiner avec une 3ème région ...)

\* Voir aide sur les API Windows CombineRgn, SetWindowRgn, CreateRectRgn, CreateRoundRectRgn, CreateEllipticRgn, CreatePolygonRgn ...

\* La partie du code qui est dans FormPaint ne sert que pour tracer la bordure elle n'est pas nécessaire pour la forme de la fenêtre

Pour ceux qui ont la chance d'avoir BCB6 et Windows 2000 ou XP, on peut faire cela beaucoup plus simplement. Il suffit de mettre la propriété TransparentColor de la Form à true, de poser un TShape ou mieux un TImage sur la Form et d'affecter la couleur du TShape ou l'une des couleurs du Bitmap du TImage à la propriété TransparentColorValue de la Form.

**Déplacement d'une fenêtre sans barre de titre :**

Le déplacement de la fenêtre (Form1) se fera par maintient du bouton gauche de la souris enfoncé sur une zone quelconque de la surface de la fenêtre.

Dans Unit1.h déclarer les variables :

private: // Déclarations de l'utilisateur

bool Deplacement;

int XPos, YPos;

Dans Unit1.cpp sur les événements OnMouseDown, OnMouseMove et OnMouseUp :

void \_\_fastcall TForm1::FormMouseDown(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y)

{

XPos = X;

YPos = Y;

if (Button == mbLeft) Deplacement = true;

}

void \_\_fastcall TForm1::FormMouseMove(TObject \*Sender, TShiftState Shift,

int X, int Y)

{

if ( Deplacement == true )

{

Left += X-XPos;

Top += Y-YPos;

}

}

void \_\_fastcall TForm1::FormMouseUp(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y)

{

Deplacement = false;

}

**Autre solution beaucoup plus simple :** Sur l'événement OnMouseDown de la fiche (Form1)

void \_\_fastcall TForm1::FormMouseDown(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y)

{

ReleaseCapture();

SendMessage(Handle, WM\_SYSCOMMAND, 0xF012, 0);

}

**Obtenir les ressources User, System et GDI :**

Exemple sur une fiche (Form1) poser un Bouton (Button1) et 3 Labels (Label1, Label2, Label3)

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

typedef LONG (\* GETRES)(int);

HINSTANCE hInst = LoadLibrary("RSRC32.dll");

if (hInst != NULL)

{

GETRES pGetRes = (GETRES) GetProcAddress(hInst,

"\_MyGetFreeSystemResources32@4");

if (pGetRes)

{

long lSysRes = pGetRes(0);

long lGdiRes = pGetRes(1);

long lUsrRes = pGetRes(2);

Label1->Caption = "System : " + String(lSysRes) + "%";

Label3->Caption = "GDI : " + String(lGdiRes) + "%";

Label2->Caption = "User : " + String(lUsrRes) + "%";

}

else ShowMessage("Erreur appel fonction");

}

else ShowMessage("Erreur chargement dll");

if (hInst) FreeLibrary(hInst);

}

Ne fonctionne pas sous Windows XP.

**Charger un fichier wav en mémoire puis le jouer:**

#include <mmsystem.h>

//.......

TMemoryStream \*MSt = new(TMemoryStream);

MSt->LoadFromFile("son.wav");

sndPlaySound((const char \*)MSt->Memory,SND\_MEMORY | SND\_ASYNC);

delete MSt;

**Son wave depuis une ressource :**

fichier.rc à ajouter au projet

TONWAVE WAVE "fichier.wav"

Unit.cpp

#include <mmsystem.h>

//.........

sndPlaySound("TONWAVE",SND\_RESOURCE | SND\_ASYNC);

**ToolBar détachable :**

**Poser un TPanel (Panel1) sur la fiche (Form1) avec les propriétés :**

\* Align = alTop

\* AutoSize = True ( A faire après avoir posé la ToolBar)

\* BevelOuter = bvNone

\* UseDockManager = False

\* DockSite = True

**Sur Panel1 poser une ToolBar (ToolBar1) avec les propriétés:**

\* BorderWidth = 1

\* DragKind = dkDock

\* DragMode = dmAutomatic

\* Indent = 2

**Evénement OnCreate de Form1(Taille de la ToolBar quand elle est flottante):**

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

ToolBar1->UndockHeight = 60;

ToolBar1->UndockWidth = 128;

}

**Evénement OnDockDrop de Panel1:**

void \_\_fastcall TForm1::Panel1DockDrop(TObject \*Sender,

TDragDockObject \*Source, int X, int Y)

{

ToolBar1->Align = alTop;

}

**Evénement OnDockOver de Panel1 (dessin du rectangle de déplacement sur Panel1):**

void \_\_fastcall TForm1::Panel1DockOver(TObject \*Sender,

TDragDockObject \*Source, int X, int Y, TDragState State,

bool &Accept)

{

TRect ARect;

ARect.Left = Panel1->ClientOrigin.x;

ARect.Top = Panel1->ClientOrigin.y;

ARect.Bottom = Panel1->ClientOrigin.y + 30; //30 hauteur

ARect.Right = Panel1->ClientOrigin.x+Panel1->Width;

Source->DockRect = ARect;

}

La ToolBar pouvant être rendu invisible avec le bouton fermeture. prévoir de la rendre à nouveau visible avec le code suivant: ToolBar1->Visible = true; (par exemple dans un menu.)

Les ControlBar permettent de faire cela beaucoup plus facilement. Cet exemple démontre plus le docking de contrôles.

**Créer un raccourci:**

**Exemple un raccourci de la calculatrice sur le bureau au click d'un bouton:**

#include <shlobj.h>

//.........

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

HRESULT hres;

IShellLink\* psl;

WORD wsz[MAX\_PATH];

CoInitialize(NULL);

hres = CoCreateInstance(CLSID\_ShellLink, NULL,CLSCTX\_INPROC\_SERVER,

IID\_IShellLink, (LPVOID \*)&psl);

if (SUCCEEDED(hres))

{

IPersistFile\* ppf;

psl->SetPath("C:\\Windows\\calc.exe");

psl->SetDescription("Calculatrice");

psl->QueryInterface(IID\_IPersistFile,(LPVOID \*)&ppf);

MultiByteToWideChar(CP\_ACP, 0,"C:\\Windows\\Bureau\\Calc.lnk", -1,

(LPWSTR)wsz, MAX\_PATH);

ppf->Save((LPWSTR)wsz, TRUE);

ppf->Release();

}

psl->Release();

CoUninitialize();

}

Pour ne pas surcharger l'exemple je n'ai pas employer les fonctions adéquates pour chercher le chemin de Windows et du bureau.

J'ai tester la compilation avec BCB6. Pour ne pas avoir les messages d'erreurs sur le fichier "shlobj.h", il faut déclarer la constante NO\_WIN32\_LEAN\_AND\_MEAN au tout début du fichier ".cpp" avant #include <vcl.h>

#define NO\_WIN32\_LEAN\_AND\_MEAN

#include <vcl.h>

#include <shlobj.h>

#pragma hdrstop

//.......

**Exemple d'utilisation de TPerformanceGraph :**

Sur une Form (Form1) poser un composant TPerformanceGraph (PerformanceGraph1) et un Timer (Timer1) avec sa propriété Interval à 100. Sur l'événement OnTimer de Timer1 mettre le code suivant:

#include <stdlib.h> //Pour la fonction rand()

//.............

void \_\_fastcall TForm1::Timer1Timer(TObject \*Sender)

{

static int x = 40;

x = x + rand()%5 - rand()%5;

PerformanceGraph1->DataPoint(clRed,x);

PerformanceGraph1->Update();

}

On peut dessiner plusieurs tracé en appelant plusieurs fois la méthode DataPoint, avec des couleurs différentes en changeant son premier argument. C'est la méthode Update qui provoque le dessin du tracé et le fait avancer d'un pas.

**Lancer l'économiseur d'écran :**

Par exemple sur l'événement OnClick d'un bouton.

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

SendMessage(Handle,WM\_SYSCOMMAND,SC\_SCREENSAVE,0);

}

Tapisser une Form avec un Bitmap :

Pour l'exemple nous allons mettre dans le même dossier que l'exécutable un fichier Bitmap nommé: "fond.bmp". Puis sur l'événement OnPaint de la Form (Form1) le code suivant:

void \_\_fastcall TForm1::FormPaint(TObject \*Sender)

{

Graphics::TBitmap \*Bitmap1 = new Graphics::TBitmap();

Bitmap1->LoadFromFile("fond.bmp");

for(int x=0 ; x < Width ; x+=Bitmap1->Width)

{

for (int y=0 ; y < Height ; y+=Bitmap1->Height)

Canvas->Draw(x,y,Bitmap1);

}

delete Bitmap1;

}

On peut mettre le fichier Bitmap dans une ressource. Cela a l'avantage de ne pas avoir de fichier à joindre avec l'application.

**Extraire les mots d'une AnsiString :**

Sur une fiche poser un TMemo (Memo1) un TEdit (Edit1) et un bouton (Button1) A l'événement OnClick de Button1 les mots de la phrase contenue dans Edit1 s'afficheront sur différentes lignes de Memo1.

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

int x;

AnsiString st = Edit1->Text;

st = st.Trim();

while ((x=st.Pos(" ")) > 0 )

{

Memo1->Lines->Add(st.SubString(1,x));

st.Delete(1,x);

st = st.TrimLeft();

}

Memo1->Lines->Add(st);

}

**Compiler une unité Delphi :**

**Pour compiler une unité Delphi :**

\* Il vous faut les fichiers ".pas" et ".dfm" par exemple "Unit1.pas" et "Unit1.dfm".

\* Vous les copiez dans le dossier où vous allez créer votre projet.

\* Puis vous créez un nouveau projet.

\* Vous retirez du projet la fiche (Form1) crée automatiquement:

MENU: "Projet" -> "Retirer du projet" -> "Unit1.cpp Form1" -> "Ok"

\* Vous ajouter au projet la fiche Delphi:

MENU: "Projet" -> "Ajouter au projet" -> "Unit1.pas" -> "Ok"

\* Si la fiche n'apparaît pas automatiquement faire "Voir une fiche"

\* Maintenant vous avez la fiche et son code source à l'écran vous pouvez compiler le projet.

Par contre, il n'est pas possible d'ajouter des composants visuellement. Il faut le faire en mode texte.

**Taille de la RAM physique :**

Obtention de la taille de la Ram en Mo

MEMORYSTATUS X;

X.dwLength = sizeof(X);

GlobalMemoryStatus(&X);

TailleDeLaRam = (X.dwTotalPhys+524288)/1024/1024;

**Mettre une commande dans le menu Système :**

Dans le fichier Unit.h on définit une constante pour la commande CM\_MONMSG. On déclare la fonction ReponseMessage et le Handle de menu MyMenu. Puis on dirige le message WM\_SYSCOMMAND vers la fonction ReponseMessage.

#define CM\_MONMSG 401

class TForm1 : public TForm

{

\_\_published: // Composants gérés par l'EDI

void \_\_fastcall FormCreate(TObject \*Sender);

private: // Déclarations de l'utilisateur

HMENU MyMenu;

void \_\_fastcall ReponseMessage(TMessage &Msg);

public: // Déclarations de l'utilisateur

\_\_fastcall TForm1(TComponent\* Owner);

BEGIN\_MESSAGE\_MAP

MESSAGE\_HANDLER(WM\_SYSCOMMAND, TMessage, ReponseMessage);

END\_MESSAGE\_MAP(TForm)

};

Dans le fichier Unit.cpp on crée la commande de menu du menu système sur l'événement OnCreate de la Form (Form1). On définit la fonction ReponseMessage

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

MyMenu = GetSystemMenu(Handle,FALSE);

AppendMenu(MyMenu,MF\_SEPARATOR ,NULL,NULL);//ajoute un séparateur

AppendMenu(MyMenu,NULL ,CM\_MONMSG , "A Propos !");

}

void \_\_fastcall TForm1::ReponseMessage(TMessage &Msg)

{

if (Msg.WParam == CM\_MONMSG)

ShowMessage("Message Reçu");

else TForm::Dispatch(&Msg);

}

**Envoyer des fichiers dans la corbeille :**

Par exemple nous allons envoyer tous les fichiers textes "\*.txt" du dossier "C:\\tmp" dans la corbeille à l'aide de la fonction SHFileOperation.

#include <shellapi.h>

//................................

SHFILEOPSTRUCT FileOpStr;

ZeroMemory(&FileOpStr, sizeof(FileOpStr));

FileOpStr.hwnd = Application->Handle;

FileOpStr.fFlags = FOF\_ALLOWUNDO;

FileOpStr.wFunc = FO\_DELETE;

FileOpStr.pFrom = "C:\\tmp\\\*.txt\0";

SHFileOperation(&FileOpStr);

Cette fonction peut aussi servir à renommer, copier, déplacer des fichiers.  
FileOpStr.wFunc: FO\_RENAME, FO\_COPY, FO\_MOVE. Dossier de destination : FileOpStr.pTo = "C:\\ici"  
Les chaînes pointées par FileOpStr.pFrom et FileOpStr.pTo peuvent contenir plusieurs fichiers ils doivent être séparer par des caractères NULL '\0' et les chaînes doivent se terminer par un double '\0'.  
Les chaînes contenant les noms de fichiers doivent contenir des chemins complets.  
  
**Exemple :**  
**Copie du dossier Source et tout son contenu dans le dossier Destination :**

#include <shellapi.h>

//................................

SHFILEOPSTRUCT FileOpStr;

ZeroMemory(&FileOpStr, sizeof(FileOpStr));

FileOpStr.hwnd = Application->Handle;

FileOpStr.wFunc = FO\_COPY;

FileOpStr.pFrom = "C:\\Source\0";

FileOpStr.pTo = "C:\\Destination\0";

SHFileOperation(&FileOpStr);

**Déplassement des fichiers du dossier Source dans le dossier Destination :**

#include <shellapi.h>

//................................

SHFILEOPSTRUCT FileOpStr;

ZeroMemory(&FileOpStr, sizeof(FileOpStr));

FileOpStr.hwnd = Application->Handle;

FileOpStr.wFunc = FO\_MOVE;

FileOpStr.pFrom = "C:\\Source\\\*.\*\0";

FileOpStr.pTo = "C:\\Destination\0";

SHFileOperation(&FileOpStr);

**Détection du changement de résolution de l'écran :**

Dans Unit.h déclaration de la méthode réagissant au message (WM\_DISPLAYCHANGE) envoyé par windows à toutes les applications quand la résolution de l'écran change.

private: // Déclarations de l'utilisateur

void \_\_fastcall ChgtResolutionEcran(TMessage &Msg);

public: // Déclarations de l'utilisateur

\_\_fastcall TForm1(TComponent\* Owner);

BEGIN\_MESSAGE\_MAP

MESSAGE\_HANDLER(WM\_DISPLAYCHANGE, TMessage, ChgtResolutionEcran);

END\_MESSAGE\_MAP(TForm)

Dans Unit.cpp définition de la fonction. C'est là que l'on met les actions à éffectuer.

void \_\_fastcall TForm1::ChgtResolutionEcran(TMessage &Msg)

{

ShowMessage("La résolution écran a changé.");

}

**Appel de la boîte de dialogue de changement d'icônes:**

Dans cet exemple nous allons appeller la boîte de dialogue de changement d'icônes. (Celle qui sert à changer les icônes des raccourcis) Puis nous afficherons l'icône sélectionnée sur la fiche.   
Sur une Form (Form1) poser un bouton (Button1) et 2 Labels (Label1 et Label2).   
  
Sur Unit.h déclaration du Handle d'icône:

private: // Déclarations de l'utilisateur

HICON MonIcone;

Sur Unit.cpp événement OnClick de Button1 et OnPaint de Form1:

#include <shellapi.h>

//................

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

typedef LONG (CALLBACK \*SHCHANGEICONDIALOG)(HWND hWnd, LPCSTR szFilename,

LONG Size, int \*IconIndex);

SHCHANGEICONDIALOG SHChangeIconDialog = NULL;

char szFilename[MAX\_PATH];

int IconIndex;

HANDLE hInst;

hInst=LoadLibrary("SHELL32.DLL");

if(hInst)

{

SHChangeIconDialog=(SHCHANGEICONDIALOG)GetProcAddress(hInst,(LPCSTR)62);

szFilename[0]='\0';

SHChangeIconDialog(Handle,szFilename,sizeof(szFilename),&IconIndex);

Label1->Caption = IconIndex;

Label2->Caption = szFilename;

MonIcone = ExtractIcon(HInstance, szFilename, IconIndex);

Refresh();

}

if(hInst) FreeLibrary(hInst);

}

void \_\_fastcall TForm1::FormPaint(TObject \*Sender)

{

DrawIcon(Canvas->Handle, 200, 10, MonIcone);

}

**Ouvrir votre application par l'ouverture de l'un de ses fichiers :**

Exemple: l'extension des fichiers de l'application exemple sera ".mon" se seront des fichiers textes qui se chargeront dans un Memo (Memo1) qui a été posé sur la fiche (Form1).  
Pour cela il suffit de mettre ce code dans l'événement OnCreate de Form1

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

if ( ParamCount() > 0 )

Memo1->Lines->LoadFromFile(ParamStr(1));

}

ParamCount() renvoie le nombre de paramètres.  
ParamStr(1) renvoie une AnsiString contenant le 1er paramètre.   
  
Mais pour que cela fonctionne, il faut associer le type de fichiers ".mon" à l'application.   
  
**On peut faire l'association par programme :**   
  
Unit.h :

#include <Registry.hpp>

Unit.cpp

TRegistry \*Reg = new TRegistry;

Reg->RootKey = HKEY\_CLASSES\_ROOT;

Reg->OpenKey("\\.mon",true);//Extention du fichier

Reg->WriteString("","Mon\_Fichier");//nom de la clé suivante

Reg->CloseKey();

Reg->OpenKey("\\Mon\_Fichier",true);

Reg->WriteString("","Fichier Mon");

//nom qui apparait dans l'explorateur de fichier

Reg->CloseKey();

Reg->OpenKey("\\Mon\_Fichier\\shell\\open\\command",true);

Reg->WriteString("","\"C:\\dossier\\mon.exe\" \"%1\"");

//Chemin et nom du programme associé suivie de "%1"

Reg->CloseKey();

delete Reg;

ATTENTION aux écritures dans la base de registre. Les erreurs ne sont pas permises.  
Ce code a été testé sous Windows 95, 98 et XP.  
Pour le tester il faudra créer un fichier texte avec l'extension ".mon"

**Sauvegarder la taille et la position de la fiche dans le registre :**

Dans cet exemple nous allons utiliser la classe **TRegistry** pour sauvegarder et restaurer la taille et la position de la dernière utilisation avant fermeture de notre fiche (Form1).  
  
  
Unit.h :

#include <Registry.hpp>

Sauvegarde: à la fermeture de la fiche sur l'événement OnClose de Form1.

void \_\_fastcall TForm1::FormClose(TObject \*Sender, TCloseAction &Action)

{

TRegistry \*Reg = new TRegistry;

Reg->OpenKey("\\Software\\Votre\_Societe\\Votre\_Application",true);

if ( WindowState == wsNormal )

{

Reg->WriteBool("Agrandie",false);

Reg->WriteInteger("Pos\_X",Left);

Reg->WriteInteger("Pos\_Y",Top);

Reg->WriteInteger("Largeur",Width);

Reg->WriteInteger("Hauteur",Height);

}

else Reg->WriteBool("Agrandie",true);

Reg->CloseKey();

delete Reg;

}

Restauration: à l'ouverture de la fiche sur l'événement OnCreate de Form1.

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

TRegistry \*Reg = new TRegistry;

if ( Reg->OpenKey("\\Software\\Votre\_Societe\\Votre\_Application",false))

{

try

{

Left = Reg->ReadInteger("Pos\_X");

Top = Reg->ReadInteger("Pos\_Y");

Width = Reg->ReadInteger("Largeur");

Height = Reg->ReadInteger("Hauteur");

}

catch(...) {}

if ( Reg->ReadBool("Agrandie") ) WindowState = wsMaximized;

}

Reg->CloseKey();

delete Reg;

}

Les données de taille et position peuvent ne pas être dans la base de registre si lors des premiéres utilisations la fiche a été fermée en position Maximised. D'où le bloc try catch pour éviter le message d'erreur en ce cas.  
La clé par défaut de TRegistry est HKEY\_CURRENT\_USER. Donc dans notre exemple les données seront écrites dans : HKEY\_CURRENT\_USER\Software\Votre\_Societe\Votre\_Application  
ATTENTION aux écritures dans la base de registre. Les erreurs ne sont pas permises.

**Sauvegarder la taille et la position de la fiche dans un fichier ini :**

Comme l'exemple précédent nous allons sauvegarder et restaurer la taille et la position de la dernière utilisation avant fermeture de notre fiche (Form1), mais cette fois dans un fichier profil ".ini" à l'aide de la classe **TIniFile**.   
  
Unit.h :

#include <Inifiles.hpp>

Sauvegarde: à la fermeture de la fiche sur l'événement OnClose de Form1.

void \_\_fastcall TForm1::FormClose(TObject \*Sender, TCloseAction &Action)

{

TIniFile \*Ini = new TIniFile("App.ini");

if ( WindowState == wsNormal )

{

Ini->WriteInteger("Position", "Pos\_X", Left);

Ini->WriteInteger("Position", "Pos\_Y", Top);

Ini->WriteInteger("Position", "Largeur", Width);

Ini->WriteInteger("Position", "Hauteur", Height);

Ini->WriteBool("Position", "Agrandie",false);

}

else Ini->WriteBool("Position", "Agrandie",true);

delete Ini;

}

Restauration: à l'ouverture de la fiche sur l'événement OnCreate de Form1.

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

TIniFile \*Ini = new TIniFile("App.ini");

Left = Ini->ReadInteger("Position", "Pos\_X", Left);

Top = Ini->ReadInteger("Position", "Pos\_Y", Top);

Width = Ini->ReadInteger("Position", "Largeur", Width);

Height = Ini->ReadInteger("Position", "Hauteur", Height);

if ( Ini->ReadBool("Position", "Agrandie", false) ) WindowState = wsMaximized;

delete Ini;

}

Utilisation des fonctions de lectures et écritures :  
Valeur = ReadInteger("Section", "Clé", ValeurParDéfaut);  
WriteInteger("Section", "Clé", Valeur);  
Dans cet exemple le fichier "App.ini" est enregistré dans le dossier Windows. Si vous voulez l'enregistrer ailleurs il faut mettre le chemin avec le nom de fichier.  
  
*Pour plus d'information voir Aide C++ Builder Rubrique* ***TIniFile****.*

**Changer le papier peint de Windows :**

Pour cela nous utiliserons la fonction **SystemParametersInfo**.

SystemParametersInfo(SPI\_SETDESKWALLPAPER,

0,"C:\\Dossier\\Dessin.bmp",NULL );

Cette fonction permet d'obtenir ou de changer de nombreux paramètres de configuration de Windows. (voir aide API Windows)  
  
**Par exemple désactiver l'écran de veille:**

SystemParametersInfo(SPI\_SETSCREENSAVEACTIVE, false, NULL, 0);

**Réactiver l'écran de veille:**

SystemParametersInfo(SPI\_SETSCREENSAVEACTIVE, true, NULL, 0);

**Une seule instance de l'application :**

Dans cet exemple le code va être inséré dans le fichier principal de notre application "Project1.cpp". Nous y créons un Mutex avec un nom qui n'a aucune chance d'exister dans une autre application. Au lancement de la deuxième instance de l'application le Mutex avec ce nom existant déjà provoque une erreur sur la fonction CreateMutex. Erreur que l'on teste pour quitter la deuxième instance de l'application le cas échéant.

#include <vcl.h>

#pragma hdrstop

USEFORM("Unit1.cpp", Form1);

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

WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

{

HANDLE hMutex;

try

{

hMutex = CreateMutex (NULL,FALSE,"Mon application");

if (GetLastError() == ERROR\_ALREADY\_EXISTS) return 0;

Application->Initialize();

Application->CreateForm(\_\_classid(TForm1), &Form1);

Application->Run();

CloseHandle(hMutex);

}

catch (Exception &exception)

{

Application->ShowException(&exception);

}

return 0;

}

**Comment gérer les événements d'un composant crée dynamiquement :**

Dans cet exemple nous allons créer un bouton dynamiquement puis créer une méthode pour gérer son événement OnClick.  
  
Dans Unit1.h déclaration du pointeur sur le bouton et de la méthode :

private: // Déclarations de l'utilisateur

TButton \*MonBouton;

void \_\_fastcall MonBoutonClick(TObject \*Sender);

Sur l'événement OnCreate de Form1 création du bouton et affectation de la méthode MonBoutonClick à l'événement OnClick du bouton :

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

MonBouton = new TButton(this);

MonBouton->Left = 20;

MonBouton->Top = 20;

MonBouton->Caption = "Bouton";

MonBouton->Parent = Form1;

MonBouton->OnClick = MonBoutonClick;

}

Définition de la méthode :

void \_\_fastcall TForm1::MonBoutonClick(TObject \*Sender)

{

ShowMessage("Tu viens de cliquer sur MonBouton");

}

**Gestion des événements OnMouseEnter et OnMouseExit :**

Pour l'exemple nous allons poser un Label (Label1) sur une fiche (Form1). Nous passerons le texte du label en rouge quand le curseur viendra au dessus puis en bleu quand il en sortira.  
  
**Unit1.h :**  
déclaration d'une variable TWndMethod pour mémoriser la procédure de fenêtre de Label1 et la méthode de la nouvelle procédure de fenêtre.

private: // Déclarations de l'utilisateur

Controls::TWndMethod OldLabelWP;

void \_\_fastcall NewLabelWP(TMessage &Msg);

**Unit1.cpp :**  
événement OnCreate de Form1, sauvegarde de l'ancienne procédure de fenêtre de Label1 et affectation de la nouvelle.

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

OldLabelWP = Label1->WindowProc;

Label1->WindowProc = NewLabelWP;

}

Nouvelle procédure de fenêtre de Label1: traitement des commandes CM\_MOUSELEAVE et CM\_MOUSEENTER sinon appel de l'ancienne procédure de fenêtre si autre message.

void \_\_fastcall TForm1::NewLabelWP(TMessage &Msg)

{

if (Msg.Msg == CM\_MOUSELEAVE) //OnMouseExit

{

Label1->Font->Color = clBlue;

}

else if (Msg.Msg == CM\_MOUSEENTER) //OnMouseEnter

{

Label1->Font->Color = clRed;

}

else OldLabelWP(Msg);

}

Cette façon de faire est obsolète sur BCB6, on a accés à ces événements dans l'inspecteur d'objets (OnMouseEnter et OnMouseLeave).

**Hint sur plusieurs lignes :**

Pour que la propriété Hint s'affiche sur plusieurs ligne. Il faut l'affecter à l'éxécution par exemple sur l'événement OnCreate de la Form.

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

Hint = "Bonjour\nMonsieur";

}

**Changer la couleur d'une ProgressBar :**

Exemple : passer la couleur de ProgressBar1 en rouge.

SendMessage(ProgressBar1->Handle, PBM\_SETBARCOLOR, 0, clRed);

**Drag and Drop de l'explorateur Windows vers votre application :**

Dans cet exemple nous allons copier les noms et chemins des fichiers glissés depuis l'explorateur Windows vers une ListBox (ListBox1) qui a été posé sur la Form (Form1).  
  
**Unit.h** déclaration de la méthode DropFiles réagissant au message WM\_DROPFILES

private: // Déclarations de l'utilisateur

void DropFiles(TMessage &Message);

public: // Déclarations de l'utilisateur

\_\_fastcall TForm1(TComponent\* Owner);

BEGIN\_MESSAGE\_MAP

MESSAGE\_HANDLER(WM\_DROPFILES, TMessage, DropFiles);

END\_MESSAGE\_MAP(TForm);

**Unit.cpp** sur l'événement OnCreate de Form1 appel de la fonction **DragAcceptFiles** pour autoriser l'application à recevoir les noms fichiers depuis l'explorateur.  
Puis définition de la méthode DropFiles dans laquelle le premier appel de la fonction **DragQueryFile** nous retourne le nombre de fichiers glissés et les appels suivants leurs noms et chemins.  
La fonction **DragFinish** libère la mémoire.

#include <shellapi.h>

//..........

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

DragAcceptFiles(Handle, true);

}

void TForm1::DropFiles(TMessage &Message)

{

int nFiles;

char buffer[256];

nFiles = DragQueryFile((HDROP)Message.WParam, 0xFFFFFFFF, NULL, 0);

for (int i = 0; i < nFiles; i++)

{

DragQueryFile((HDROP)Message.WParam, i, buffer, 256);

ListBox1->Items->Add((AnsiString)buffer);

}

DragFinish((HDROP)Message.WParam);

}

**Réduire toutes les fenêtres du bureau :**

Pour cela il suffit d'envoyer la commande 415 à la barre des taches.

PostMessage(FindWindow("Shell\_TrayWnd", NULL), WM\_COMMAND, 415, 0);

D'autre commandes sont possible :  
  
- 403 Les fenêtres en cascade.  
- 404 Les fenêtres en mosaique horizontale.  
- 405 Les fenêtres en mosaique verticale.  
- 416 Annule la dernière action parmis les précedentes.  
- 413 Appelle la fenêtre de propriété de la barre des taches.  
- 408 Appelle la fenêtre de propriété date heure.  
- 401 Appelle La fenêtre Executer.

**Changer les couleurs System :**

Dans cet exemple nous allons passer la couleur du fond du bureau en rouge à l'aide de la fonction SetSysColors().

int elem = COLOR\_DESKTOP;

COLORREF color = RGB(255,0,0);

SetSysColors(1,&elem,&color);

Cette fonction doit recevoir en paramétres le nombre d'éléments à changer, puis l'adresse du tableau éléments, puis l'adresse du tableau de COLORREF qui contient les couleurs.  
Donc si plusieurs éléments notre exemple devient:

int elem[2];

COLORREF color[2];

elem[0] = COLOR\_DESKTOP;

elem[1] = COLOR\_ACTIVECAPTION;

color[0] = RGB(255,0,0);

color[1] = RGB(0,0,255);

SetSysColors(2,elem,color);

Liste des identificateurs d'élément:  
COLOR\_3DDKSHADOW, COLOR\_3DFACE, COLOR\_BTNFACE, COLOR\_3DHILIGHT, COLOR\_3DLIGHT, COLOR\_3DSHADOW, COLOR\_ACTIVEBORDER, COLOR\_ACTIVECAPTION,COLOR\_APPWORKSPACE, COLOR\_BACKGROUND, COLOR\_DESKTOP, COLOR\_BTNTEXT, COLOR\_CAPTIONTEXT, COLOR\_GRAYTEXT COLOR\_HIGHLIGHT, COLOR\_HIGHLIGHTTEXT, COLOR\_INACTIVEBORDER, COLOR\_INACTIVECAPTION, COLOR\_INACTIVECAPTIONTEXT, COLOR\_INFOBK, COLOR\_INFOTEXT, COLOR\_MENU, COLOR\_MENUTEXT, COLOR\_SCROLLBAR, COLOR\_WINDOW, COLOR\_WINDOWFRAME, COLOR\_WINDOWTEXT.

**Griser le bouton fermeture et interdire le <Alt F4> :**

Pour cela il suffit de rajouter le style CS\_NOCLOSE à la classe de fenêtre de la Form.  
  
Unit1.h

void \_\_fastcall CreateParams(TCreateParams& Params);

Unit1.cpp

void \_\_fastcall TForm1::CreateParams(TCreateParams& Params)

{

TForm::CreateParams(Params);

Params.WindowClass.style |= CS\_NOCLOSE;

}

**Cacher l'application de la barre des tâches :**

Pour cela il suffit de donner le style WS\_EX\_TOOLWINDOW à la fenêtre Application.  
(Sous BCB la fenêtre Application est toujours invisible)   
Dans le WinMain fichier: Project1.cpp

WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

{

try

{

SetWindowLong(Application->Handle, GWL\_EXSTYLE, WS\_EX\_TOOLWINDOW);

Application->Initialize();

//.........

**Lancement de votre application au démarrage de l'ordinateur :**

Inscription dans le registre du lancement automatique de votre application au démarrage de l'ordinateur.   
  
Dans Unit1.h

#include <Registry.hpp>

Dans Unit1.cpp

TRegistry \*Reg = new TRegistry();

Reg->RootKey = HKEY\_LOCAL\_MACHINE;

Reg->OpenKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", false);

Reg->WriteString("Nom du Programme", "C:\\Dossier\\Project1.exe");

delete Reg;

**Désactiver les touches Windows (Win 95) :**

Désactiver les touches Win, Ctrl-Echap , Alt-Tab et Ctrl-Alt-Suppr pendant l'éxécution de votre programme. (Ce code ne fonctionne que sous les plateformes Windows 95.)  
  
Sur les événements OnCreate et OnClose de Form1 :

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

int val;

SystemParametersInfo( SPI\_SCREENSAVERRUNNING, 1, &val, 0);

}

void \_\_fastcall TForm1::FormClose(TObject \*Sender, TCloseAction &Action)

{

int val;

SystemParametersInfo( SPI\_SCREENSAVERRUNNING, 0, &val, 0);

}

**Mettre une copie d'écran dans le presse papier:**

On va tout simplement simuler l'appuit sur la touche <Impr écran> à l'aide de la fonction **keybd\_event**

keybd\_event(VK\_SNAPSHOT,1,0,0);

Par la même méthode on peut faire une copie de la Form dans le presse papier:

keybd\_event(VK\_SNAPSHOT,0,0,0);

**Liste des fontes du système dans une Combobox :**

Sur une Form (Form1) poser une ComboBox (ComboBox1) avec sa propriété Style à csDropDownList. Sur l'événement OnCreate de Form1 affecter la liste des fontes à la propriété Items de ComboBox1 et sélectionner la fonte "Arial" :

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

ComboBox1->Items = Screen->Fonts;

ComboBox1->ItemIndex = ComboBox1->Items->IndexOf("Arial");

}

**Avoir un bouton sur la barre des tâches pour les fiches secondaires :**

Exemple pour Form2 sur son l'événement OnCreate, lui donner le style étendu WS\_EX\_APPWINDOW.

void \_\_fastcall TForm2::FormCreate(TObject \*Sender)

{

SetWindowLong(Handle,GWL\_EXSTYLE,GetWindowLong(Handle,GWL\_EXSTYLE) |

WS\_EX\_APPWINDOW);

}

**Obtenir le numéro de série d'un dique :**

Pour l'exemple sur une fiche Form1, poser un bouton Button1 et 3 labels Label1, Label2 et Label3. Puis sur l'événement OnClick de Button1 inserer le code ci-dessous. C'est la fonction de l'API Windows GetVolumeInformation qui va nous fournir ces informations. Le numéro de série du volume est retourné dans une variable de type DWORD dont on passe l'adresse au 4eme paramètre de la fonction. Dans l'exemple il est affiché sur Label1 en hexadécimal. Cette fonction peut renvoyer aussi le système de fichier du volume, affiché par Label2 dans l'exemple et aussi le nom du volume sur Label3 dans l'exemple. Il suffit de lui fournir l'adresse et la taille des buffers ou elle doit écrire ces informations.

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

DWORD NumSerie;

char NomVol[12];

char SysFile[16];

GetVolumeInformation("C:/", NomVol, 12, &NumSerie, NULL , NULL, SysFile, 16);

Label1->Caption = IntToHex((int)NumSerie, 8);

Label2->Caption = SysFile;

Label3->Caption = NomVol;

}

**Liste des disques dans une TStrings :**

Fonction retournant la liste des disques dans une liste de chaines (TStrings). Pour obtenir cette liste nous utiliserons la fonction **GetLogicalDriveStrings** de l'API Windows. Cette fonction retourne cette liste dans un buffer (DriveSt dans l'exemple), où chaque chaine est separée par un \0 et la fin de liste par un double \0.

void DiskList(TStrings\* Liste)

{

int ch = 0;

char DriveSt[127];

GetLogicalDriveStrings(126, DriveSt);

while ( DriveSt[ch] != 0 )

{

Liste->Add(&DriveSt[ch]);

ch += strlen(&DriveSt[ch])+1;

}

}

Exemple d'utilisation de cette fonction pour initialiser un TMemo avec cette liste :

DiskList(Memo1->Lines);

**Obtenir la taille d'un fichier :**

Pour l'exemple nous metterons le code sur l'événement OnClick d'un Button (Button1), le résultat sur un Label (Label1) et l'information sera obtenu avec la fonction GetFileSize. (Elle retourne -1 si le fichier n'existe pas.)

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

HANDLE hFile;

hFile = CreateFile("fichier.ext", 0, 0, NULL, OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL, NULL);

int dwSize = GetFileSize (hFile, NULL);

CloseHandle(hFile);

Label1->Caption = dwSize;

}

**Mettre une image dans les ressources d'une application :**

Dans cet exemple on va utiliser une image que l'on va inclure dans les ressources de l'application. C'est-à-dire qu'elle sera intégrée au fichier executable de l'application une fois celui-ci crée. Pour cela il faudra tout d'abord mettre le fichier de l'image (image.bmp) dans le dossier de compilation. Puis créer un fichier script de ressource qui devra être ajouté au projet de l'application.  
Image.rc pour l'exemple :

IMAGE BITMAP image.bmp

Dans ce script le nom de la ressource sera IMAGE, elle est de type BITMAP est se trouve dans le fichier image.bmp  
  
Nous allons voir maintenant comment exploiter cette ressource :  
Dans l'exemple nous allons récuperer l'image dans un TBitmap à l'aide de sa méthode **LoadFromResourceName** puis la dessiner sur la fiche (Form1), cela sur l'événement OnPaint de la fiche :

void \_\_fastcall TForm1::FormPaint(TObject \*Sender)

{

Graphics::TBitmap \*Image = new Graphics::TBitmap();

Image->LoadFromResourceName((int)HInstance, "IMAGE");

Canvas->Draw(0, 0,Image);

delete Image;

}

**Mettre l'économiseur d'écran dans un TPanel :**

Les économiseurs d'écrans sont des fichiers exécutables spéciaux. Pour les lancer, on les appelle avec le paramètre '/p' et pour obtenir leur boîte de dialogue de configuration on les appelle le paramètre '/c'. On peut aussi les lancer avec comme 2ème paramètre un Handle de fenêtre après le paramètre '/p', indiquant la fenêtre où il devra s'afficher. Dans notre exemple nous fournirons donc le Handle d'un TPanel (Panel1) afin d'afficher l'économiseur d'écran à l'intérieur de ce TPanel.

HANDLE hProcess;

DWORD Code;

//........

// Lancement de l'économiseur :

AnsiString st = "economiseur.scr /p" + IntToStr((int)Panel1->Handle) ;

STARTUPINFO StUpInfo;

PROCESS\_INFORMATION ProcessInfo;

StUpInfo.cb = sizeof(StUpInfo);

StUpInfo.lpReserved = NULL;

StUpInfo.lpDesktop = NULL;

StUpInfo.lpTitle = NULL;

StUpInfo.dwFlags = 0;

StUpInfo.cbReserved2 = 0;

StUpInfo.lpReserved2 = NULL;

CreateProcess(NULL, st.c\_str(), NULL, NULL, false, DETACHED\_PROCESS,

NULL, NULL, &StUpInfo, &ProcessInfo);

hProcess = ProcessInfo.hProcess;

GetExitCodeProcess(hProcess,&Code);

//........

//Arrêt de l'économiseur :

TerminateProcess(hProcess,Code);

L'économiseur est lancé avec la fonction CreateProcess afin de pouvoir récupérer le Handle de son process dans la structure PROCESS\_INFORMATION afin de pouvoir l'arrêter ultérieurement à l'aide de la fonction TerminateProcess. Cet arrêt est nécessaire avant la destruction de la fenêtre parente.

**Cacher la fenêtre principal au lancement de l'application :**

Il faut passer la propriété **ShowMainForm** de l'objet Application à false avant d'appeler sa méthode Run. Il faut bien sûr laisser la propriété Visible de cette fiche à false.

WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

{

try

{

Application->Initialize();

Application->ShowMainForm = false;

Application->CreateForm(\_\_classid(TForm1), &Form1);

Application->Run();

}

//...

**harger un bitmap depuis une dll :**

Dans cet exemple nous allons charger le bitmap représentant le roi de carreau depuis cards.dll, librairie contenant les cartes du fameux solitaire.  
Sur une fiche Form1 poser un TButton Button1 et une TImage Image1. Puis mettre le code suivant sur l'événement OnClick de Button1.

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

HINSTANCE HInstCard = LoadLibrary("cards.dll");

Image1->Picture->Bitmap->LoadFromResourceID((unsigned int)HInstCard,26);

FreeLibrary(HInstCard);

}

**Ajouter une console à votre programme :**

Nous allons voir à l'aide d'un exemple simple comment ajouter une console à un programme fenêtré. Pour l'exemple nous utiliserons une fiche muni de deux boutons, qui servirons à afficher deux messages différant dans la console. L'initialisation de la console se fera sur l'événement OnCreate de la fiche et sa libération sur l'événement OnDestroy de la fiche.

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"

*//---------------------------------------------------------------------------*

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

*//---------------------------------------------------------------------------*

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner)

{

}

*//---------------------------------------------------------------------------*

**void** \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

AllocConsole();

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),

FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE );

}

*//---------------------------------------------------------------------------*

**void** \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

WriteConsole(GetStdHandle(STD\_OUTPUT\_HANDLE),

"Vulguaire message.\n",19,NULL,NULL);

}

*//---------------------------------------------------------------------------*

**void** \_\_fastcall TForm1::Button2Click(TObject \*Sender)

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),

FOREGROUND\_RED | FOREGROUND\_INTENSITY);

WriteConsole(GetStdHandle(STD\_OUTPUT\_HANDLE),

"Message important.\n",19,NULL,NULL);

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),

FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE );

}

*//---------------------------------------------------------------------------*

**void** \_\_fastcall TForm1::FormDestroy(TObject \*Sender)

{

FreeConsole();

}

**TRichEdit**

**Exemple d'utilisation d'un TRichEdit : (Micro traitement de textes)**

Sur une Form (Form1) poser :

* Un Panel (Panel1) avec les propriétés :

Align à alTop  
Caption à (vide)

* Un TRichEdit (RichEdit1) avec la propriété : Align à alClient
* Une TFontDialog (FontDialog1).

Sur Panel1 poser :

* Un bouton (Button1) avec la propriété : Caption => "Police"
* Deux CheckBox :

CheckBox1 avec la propriété : Caption à "Gras"  
CheckBox2 avec la propriété : Caption à "Italique"

* Une ComboBox (ComboBox1) avec sa propriété Items remplie avec une série de valeurs de tailles de Fontes (valeurs numériques entières).

Sur l'événement **OnClick** de Button1 : appel de la boîte de dialogue de changement de fonte préalablement affecté avec la fonte de la position du curseur (caret). Puis changement de la fonte de la sélection ou de la position du caret. Puis redonner le focus au RichEdit.

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

FontDialog1->Font->Assign(RichEdit1->SelAttributes);

if (FontDialog1->Execute())

{

RichEdit1->SelAttributes->Assign(FontDialog1->Font);

}

RichEdit1SelectionChange(NULL); //voir plus loin dans ce document.

RichEdit1->SetFocus();

}

Sur l'événement **OnClick** de CheckBox1 : Mise de la fonte en Gras ou non selon que la case est cochée ou pas. Puis redonner le focus au RichEdit.

void \_\_fastcall TForm1::CheckBox1Click(TObject \*Sender)

{

if(CheckBox1->Checked)

RichEdit1->SelAttributes->Style =

RichEdit1->SelAttributes->Style << fsBold;

else

RichEdit1->SelAttributes->Style =

RichEdit1->SelAttributes->Style >> fsBold;

RichEdit1->SetFocus();

}

Sur l'événement **OnClick** de CheckBox2 : Mise de la fonte en Italique ou non selon que la case est cochée ou pas. Puis redonner le focus au RichEdit.

void \_\_fastcall TForm1::CheckBox2Click(TObject \*Sender)

{

if(CheckBox2->Checked)

RichEdit1->SelAttributes->Style =

RichEdit1->SelAttributes->Style << fsItalic;

else

RichEdit1->SelAttributes->Style =

RichEdit1->SelAttributes->Style >> fsItalic;

RichEdit1->SetFocus();

}

Sur l'événement **OnChange** de ComboBox1 : Attribution de la nouvelle taille de fonte.  
Sur l'événement **OnClick** de ComboBox1 : Redonner le focus au RichEdit.  
Et sur l'événement **OnKeyPress** de ComboBox1 : Redonner le focus au RichEdit si touche "Entrée" enfoncée et interdire les caractères non numériques.

void \_\_fastcall TForm1::ComboBox1Change(TObject \*Sender)

{

RichEdit1->SelAttributes->Size = StrToInt(ComboBox1->Text);

}

void \_\_fastcall TForm1::ComboBox1Click(TObject \*Sender)

{

RichEdit1->SetFocus();

}

void \_\_fastcall TForm1::ComboBox1KeyPress(TObject \*Sender, char &Key)

{

if (Key == 13) RichEdit1->SetFocus();

if ((Key < '0' || Key >'9') && Key != 8) Key = NULL;

}

Et sur l'événement **OnSelectionChange** de RichEdit1 donner aux contrôles permettant de modifier les propriétés du texte la valeur correspondant à la propriété du texte de la position en cours du curseur (caret).

void \_\_fastcall TForm1::RichEdit1SelectionChange(TObject \*Sender)

{

if(RichEdit1->SelAttributes->Style.Contains(fsBold))

CheckBox1->Checked = true;

else

CheckBox1->Checked = false;

if(RichEdit1->SelAttributes->Style.Contains(fsItalic))

CheckBox2->Checked = true;

else

CheckBox2->Checked = false;

ComboBox1->Text = RichEdit1->SelAttributes->Size;

}

Voir aide C++ Builder sur les formats de texte rubrique **TTextAttributes**.  
Pour utiliser la touche de tabulation sur le RichEdit il faut mettre sa propriété **WantTabs** à true.  
  
J'ai volontairement utilisé des contrôles simples pour ne pas surcharger l'exemple et ce concentrer sur le composant TRichEdit.

**Sauvegarde et Impression avec un RichEdit**

Pour imprimer le contenu d'un RichEdit il faut utiliser la méthode **Print** avec en paramètre une AnsiString contenant le titre de l'impression.

RichEdit1->Print("Titre du document");

La sauvegarde et le chargement de fichiers sont tout aussi simple.  
*Laisser la Propriété PlainText du RichEdit à false pour garder le format du texte lors de la sauvegarde ou du chargement du fichier.*

RichEdit1->Lines->SaveToFile("Fichier.rtf"); //Sauvegarge.

RichEdit1->Lines->LoadFromFile("Fichier.rtf"); //Chargement

**Formats de paragraphes**

Alignements de paragraphes : Le paragraphe qui contient le curseur (Caret) sera aligné à droite à gauche ou centré en fonction de la ligne de code suivante qui sera exécutée.

//Alignement à gauche

RichEdit1->Paragraph->Alignment = taLeftJustify ;

//Alignement au centre

RichEdit1->Paragraph->Alignment = taCenter;

//Alignement à droite

RichEdit1->Paragraph->Alignment = taRightJustify;

Voir aide C++ Builder rubrique **TParaAttributes** pour les autres possibilités de format de paragraphe. (retrait 1ère ligne, retrait à gauche, retrait à droite, les bulets, et les tabulations)

**Insertion de texte formaté**

Nous allons insérer du texte formaté à la position du curseur (Caret). Dans cet exemple nous changerons sa couleur.

RichEdit1->SelAttributes->Color = clRed;

RichEdit1->SelText = " Rouge";

RichEdit1->SelAttributes->Color = clGreen;

RichEdit1->SelText = " Vert";

RichEdit1->SelAttributes->Color = clBlue;

RichEdit1->SelText = " Bleu";

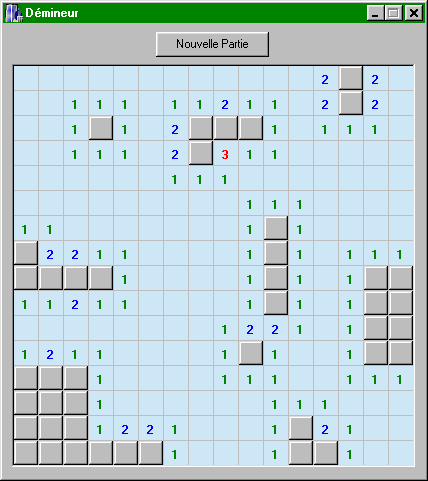
RichEdit1->SelText = "\r\n"; //Ligne suivante

La position du curseur peut être modifiée avec la propriété **SelStart** du RichEdit.  
  
Dans ce document je vous ai parlé des principales propriétés d'un RichEdit. Pour plus de détail voir aide C++ Builder. Mais vous avez déjà matière à vous entraîner à sa manipulation.  
  
RichEditement votre!

**Un démineur dans une TDrawGrid.**

**Exemple d'utilisation d'une TDrawGrid : ( Un démineur )**

Pour notre exemple nous allons construire un jeu connu de tous, "le démineur". Dans l'exemple il aura 16 x 16 cases et 20 mines. Nous utiliserons pour cela une DrawGrid où nous dessinerons dans chaque case leur valeur. (Cachée, Mine, Valeur, Vide.)

  
Sur une Form (Form1) poser :

* Une DrawGrid (DrawGrid1) avec les propriétés :

Height à 403  
Width à 403  
DefaultColWidth à 24  
DefaultRawWidth à 24  
ColCount à 16  
RawCount à 16   
FixedCols à 0  
FixedRows à 0  
ScrollBars à ssNone

* Un Bouton (Button1) avec la propriété : Caption à "Nouvelle partie"

**Unit1.h :**  
Déclaration des données membre et méthodes.  
La DrawGrid n'ayant pas d'emplacement pour stocker les valeurs de ses cases, nous les stockerons donc dans un tableau.(TabVal[16][16])  
Le 2eme tableau (TabCase[16][16]) nous indiquera si la case est : cachée, visible ou en cours de traitement des cases vide adjacentes.

private: // Déclarations de l'utilisateur

char TabCase[16][16];

char TabVal[16][16];

int NbMine;

bool PasFini;

void \_\_fastcall Initialisation();

void \_\_fastcall Decouvre(int x,int y);

**Unit1.cpp :**  
Sur l'événement **OnCreate** de Form1  
Affectation du nombre de mines à NbMine et Initialisation.

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

NbMine = 20;

Initialisation();

}

Définition de la méthode **Initialisation**  
C'est ici que l'on rempli le tableau avec les mines et que l'on calcule la valeur des cases. Puis rafraîchissement de la DrawGrid par sa méthode **Invalidate()**.

void \_\_fastcall TForm1::Initialisation()

{

int x, y;

PasFini = true ;

ZeroMemory(TabCase, sizeof(TabCase));

//Poser les mines aléatoirement

ZeroMemory(TabVal, sizeof(TabVal));

randomize();

for (int z=0 ; z < NbMine ; z++)

{

x = rand() % 16 ;

y = rand() % 16 ;

TabVal[x][y] = 10;

}

//Remplir le tableau de valeur

for (x=0 ; x < 16; x++)

for (y=0 ; y < 16; y++)

{

if (TabVal[x][y] >= 10)

{

if(y<=14)TabVal[x][y+1]++;

if(y>0)TabVal[x][y-1]++;

if(x<=14)TabVal[x+1][y]++;

if(x<=14 && y<=14)TabVal[x+1][y+1]++;

if(x<=14 && y>0)TabVal[x+1][y-1]++;

if(x>0)TabVal[x-1][y]++;

if(x>0 && y<=14)TabVal[x-1][y+1]++;

if(x>0 && y>0)TabVal[x-1][y-1]++;

}

}

//Rafraîchir la grille

DrawGrid1->Invalidate();

}

Sur l'événement **OnClick** de Button1. Initialisation pour une nouvelle partie.

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

Initialisation();

}

Evénement **OnDrawCell** de DrawGrid1:  
C'est ici que l'on dessine. Cette méthode est appelée pour chaque case. La case en cours de traitement est reçue pas les paramètres **ACol** et **ARow** et sa position et ses dimensions par le paramètre **Rect**. Le paramètre **State** renvoie son état, par exemple si elle a le focus. (Voir aide BCB rubrique TGridDrawState.)   
  
Pour dessiner nous nous servirons du **Canvas** de la DrawGrid. Nous passerons sa fonte en Gras, la couleur de sa Brush en clBtnFace pour les cases cachées et en clWindow pour les cases visibles, qui nous servira à rempir les cases avec la méthode **FillRect**. Nous utiliserons la fonction de l'API Windows **DrawEdge** pour dessiner le relief des cases cachées. Pour le dessin des cases visible nous utiliserons la méthode **TextRect** du Canvas. Elle a l'avantage par rapport à **TextOut** de ne pas déborder de la case. (Dans cet exemple on aurait pu utiliser TextOut.) Puis on fini par effacer le rectangle de focus avec la méthode **DrawFocusRect** du canvas. (Pour effacer un rectangle de focus il suffit de le dessiner une seconde fois.)

void \_\_fastcall TForm1::DrawGrid1DrawCell(TObject \*Sender, int ACol,

int ARow, TRect &Rect, TGridDrawState State)

{

//Texte en Gras

DrawGrid1->Canvas->Font->Style = DrawGrid1->Canvas->Font->Style << fsBold;

if(TabCase[ACol][ARow]==0) //Dessiner les cases cachées.

{

DrawGrid1->Canvas->Brush->Color = clBtnFace;

DrawGrid1->Canvas->FillRect(Rect);

DrawEdge(DrawGrid1->Canvas->Handle,&Rect,EDGE\_RAISED,BF\_RECT);

}

else //Dessiner les cases découvertes.

{

DrawGrid1->Canvas->Brush->Color = clWindow;

DrawGrid1->Canvas->FillRect(Rect);

if (TabVal[ACol][ARow] >= 10) DrawGrid1->Canvas->

TextRect(Rect, Rect.Left+7, Rect.Top+7, "M");

if(TabVal[ACol][ARow]==1) DrawGrid1->Canvas->Font->Color = clGreen;

if(TabVal[ACol][ARow]==2) DrawGrid1->Canvas->Font->Color = clBlue;

if(TabVal[ACol][ARow]>2 && TabVal[ACol][ARow]<9)

DrawGrid1->Canvas->Font->Color = clRed;

if (TabVal[ACol][ARow] >= 0 && TabVal[ACol][ARow] < 9) DrawGrid1->Canvas->

TextRect(Rect, Rect.Left+7, Rect.Top+7,

IntToStr(TabVal[ACol][ARow]));

if (TabVal[ACol][ARow] == 0 ) DrawGrid1->Canvas->

TextRect(Rect, Rect.Left+7, Rect.Top+7, "");

//Cette ligne n'écris rien, mais sans elle il y a des problémes

//pour éffacer le rectangle de focus.

}

//Effacer le rectangle de focus.

if(State.Contains(gdFocused)) DrawGrid1->Canvas->DrawFocusRect(Rect);

}

Sur l'événement **OnClick** de DrawGrid1 on récupère la case cliquée dans les propriétés **Col** et **Row** de la DrawGrid. Elles indiquent la case sélectionnée, et comme en cliquant sur la case on l'a sélectionnée, alors le tour est joué. Le reste du code sert pour le traitement du jeu. Nous utilisons la méthode **Invalidate()** de la DrawGrid là aussi pour la rafraîchir.

void \_\_fastcall TForm1::DrawGrid1Click(TObject \*Sender)

{

if (PasFini == false) return;

bool fin=true;

int x=DrawGrid1->Col;

int y=DrawGrid1->Row;

TabCase[x][y]=1 ;

//Tout découvrir si mine

if (TabVal[x][y] >=10)

{

for (int x=0 ; x < 16; x++)

for (int y=0 ; y < 16; y++)

TabCase[x][y]=1 ;

PasFini = false;

DrawGrid1->Invalidate();

ShowMessage("Vous avez perdu");

return;

}

//Découvrir les case vides et adjacente

if (TabVal[x][y] == 0)

{

Decouvre(x,y);

}

do

{

fin = false;

x=0;

while (x<=15)

{

y=0;

while (y<=15)

{

if (TabCase[x][y] == 2)

{

TabCase[x][y] = 1;

Decouvre(x,y);

fin = true;

}

y++;

}

x++;

}

}

while (fin);

//Rafréchir la grille

DrawGrid1->Invalidate();

//Gagné

PasFini=false;

for (int x=0 ; x < 16; x++)

for (int y=0 ; y < 16; y++)

if(TabCase[x][y]==0 && TabVal[x][y]<10) PasFini = true ;

if (PasFini == false) ShowMessage("Vous avez gagné");

}

Définition de la méthode **Decouvre** qui sert aussi au traitement du jeu.

void \_\_fastcall TForm1::Decouvre(int x,int y)

{

if(y<=14)

{

if (TabVal[x][y+1]>0 || TabCase[x][y+1]==1) TabCase[x][y+1]=1;

else TabCase[x][y+1]=2;

}

if(y>0)

{

if (TabVal[x][y-1]>0 || TabCase[x][y-1]==1) TabCase[x][y-1]=1;

else TabCase[x][y-1]=2;

}

if(x<=14)

{

if (TabVal[x+1][y]>0 || TabCase[x+1][y]==1) TabCase[x+1][y]=1;

else TabCase[x+1][y]=2;

}

if(x<=14 && y<=14)

{

if (TabVal[x+1][y+1]>0 || TabCase[x+1][y+1]==1) TabCase[x+1][y+1]=1;

else TabCase[x+1][y+1]=2;

}

if(x<=14 && y>0)

{

if (TabVal[x+1][y-1]>0 || TabCase[x+1][y-1]==1) TabCase[x+1][y-1]=1;

else TabCase[x+1][y-1]=2;

}

if(x>0)

{

if (TabVal[x-1][y]>0 || TabCase[x-1][y]==1) TabCase[x-1][y]=1;

else TabCase[x-1][y]=2;

}

if(x>0 && y<=14)

{

if (TabVal[x-1][y+1]>0 || TabCase[x-1][y+1]==1) TabCase[x-1][y+1]=1;

else TabCase[x-1][y+1]=2;

}

if(x>0 && y>0)

{

if (TabVal[x-1][y-1]>0 || TabCase[x-1][y-1]==1) TabCase[x-1][y-1]=1;

else TabCase[x-1][y-1]=2;

}

}

Dans cette application la rotation de la roulette de la souris n'est pas souhaitée. Alors pour l'invalider nous allons rajouter ces lignes de code dans les événements **OnMouseWheelUp** et **OnMouseWheelDown**.

void \_\_fastcall TForm1::DrawGrid1MouseWheelDown(TObject \*Sender,

TShiftState Shift, TPoint &MousePos, bool &Handled)

{

Handled = true;

}

void \_\_fastcall TForm1::DrawGrid1MouseWheelUp(TObject \*Sender,

TShiftState Shift, TPoint &MousePos, bool &Handled)

{

Handled = true;

}

Le code précédant a été élaboré sous BCB6.  
Aprés un test sous BCB4 il a fallu rajouter la ligne suivante dans Unit1.cpp (Pour randomize et rand.)

#include <stdlib.h>

Dans cet exemple je n'ai pas commenté la partie traitant du jeu (ce n'était pas le but de ce tutoriel), mais seulement celle traitant de la TDrawGrid.  
  
Je vous souhaite bon amusement. Et si c'est trop facile rajoutez des mines.

**Construction d'une application MDI**

|  |
| --- |
| **Sommaire :**         - [Introduction.](http://chgi.developpez.com/mdi/#p1)        - [Construction de la fiche enfant MDI.](http://chgi.developpez.com/mdi/#p2)        - [Construction de la fiche parent MDI.](http://chgi.developpez.com/mdi/#p3)        - [Questions diverses.](http://chgi.developpez.com/mdi/#p4) |

|  |
| --- |
| **Introduction :** Dans cet exemple, nous allons suivre pas à pas la construction d'une application MDI. Notre application ne fera rien de particulier si ce n'est de gérer ses fenêtres enfants. Si vous suivez ce tutoriel pas à pas vous devrez obtenir une application ressemblant à la capture d'écran ci-dessous.  http://chgi.developpez.com/mdi/mdi.gif  Tout d'abord il faut créer une nouvelle application, nous reviendrons plus tard sur sa fiche principale Form1. Construisons en premier lieu le modèle de la fiche enfant (Form2).  **Construction de la fiche enfant MDI :**  Création de la fiche enfant (Form2) : sur le menu faire : "Fichier" puis "Nouveau" puis "Fiche", mettre sa propriété **FormStyle** à fsMDIChild.  Cette fiche ne doit pas être créée automatiquement au démarrage de l'application, nous devons donc la passer en fiche disponible : Sur le menu faire "Projet" puis "Options..." puis sur la boîte de dialogue sélectionner l'onglet "Fiches" puis dans la liste de gauche (Fiches créées automatiquement) sélectionner Form2 appuyer sur le bouton ">", Form2 doit passer sur la liste de droite (Fiches disponibles). Si vous consultez le code source de project1.cpp vous remarquerez que cette ligne : *Application->CreateForm(\_\_classid(TForm2), &Form2);* a disparu. Supprimer cette ligne manuellement donne donc le même résultat. Pour agrémenter cette fiche, nous y placerons un TMemo avec sa propriété Align à alClient. Nous allons maintenant aborder le code de la fiche enfant  **Unit2.h** : Nous y déclarons une donnée membre **static** "Numero", elle existera donc en un seul exemplaire quelque soit le nombre d'instances de la classe qui seront créées. Elle nous servira justement à compter ces instances pour leur donner un numéro que l'on affichera sur leur "Caption"  #define Unit2H  //---------------------------------------------------------------------------  #include <Classes.hpp>  #include <Controls.hpp>  #include <StdCtrls.hpp>  #include <Forms.hpp>  //---------------------------------------------------------------------------  class TForm2 : public TForm  {  \_\_published: // Composants gérés par l'EDI  TMemo \*Memo1;  void \_\_fastcall FormClose(TObject \*Sender, TCloseAction &Action);  void \_\_fastcall FormCreate(TObject \*Sender);  private: // Déclarations de l'utilisateur  static int Numero;  public: // Déclarations de l'utilisateur  \_\_fastcall TForm2(TComponent\* Owner);  };  //---------------------------------------------------------------------------  extern PACKAGE TForm2 \*Form2;  //---------------------------------------------------------------------------  #endif  **Unit2.cpp :** Dans le constructeur on incrémente la donnée membre static "Numero" à chaque fois que l'on créera une nouvelle fiche enfant et on ajoutera ce numéro à son "Caption" sur l'événement OnCreate de la fiche. On pourait aussi par exemple l'affecter à la propriété tag de la fiche pour avoir un numéro unique de chaque instance dans cette propriété. Dans l'événement OnClose de la fiche, on ajoutera Action = caFree; pour détruire la fiche. (Une fiche enfant MDI est minimisée par défaut sur l'événement OnClose)  #include <vcl.h>  #pragma hdrstop  #include "Unit2.h"  //---------------------------------------------------------------------------  #pragma package(smart\_init)  #pragma resource "\*.dfm"  TForm2 \*Form2;  int TForm2::Numero = 0;  //---------------------------------------------------------------------------  \_\_fastcall TForm2::TForm2(TComponent\* Owner)  : TForm(Owner)  {  Numero++;  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm2::FormClose(TObject \*Sender, TCloseAction &Action)  {  Action = caFree;  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm2::FormCreate(TObject \*Sender)  {  Caption = "Fenêtre N° " + IntToStr(Numero);  }    **Construction de la fiche parent MDI :**  Nous allons d'abord donner les propriétés suivantes à sa Form (Form1) :    **FormStyle** à fsMDIForm    **Caption** à Application MDI  Puis poser sur la fiche un MainMenu (MainMenu1) avec un premier menu nommé "Fichier" ayant deux options de menu, "New" et "Quitter" puis un second menu nommé "Fenêtre" avec trois options "Cascade", Horizontale" et "Verticale".  **Unit1.cpp** : Tout d'abord inclure Unit2.h pour avoir accés au model de la fiche enfant. Puis sur l'événement OnClick de l'option de menu "New" nous créons dynamiquement une fiche enfant (Form2). Sur l'événement OnClick de "Quitter" nous fermons l'application. Sur l'événement OnClick de "Cascade" nous mettons les fenêtres enfants en cascade avec la méthode **Cascade**. Sur les événements OnClick de "Horizontale" et "Verticale" nous mettons les fenêtres enfants en mosaïque horizontale ou verticale à l'aide de la méthode **Tile** en fonction de l'état de la propriété **TileMode**.  #include <vcl.h>  #pragma hdrstop  #include "Unit1.h"  #include "Unit2.h"  //---------------------------------------------------------------------------  #pragma package(smart\_init)  #pragma resource "\*.dfm"  TForm1 \*Form1;  //---------------------------------------------------------------------------  \_\_fastcall TForm1::TForm1(TComponent\* Owner)  : TForm(Owner)  {  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::New1Click(TObject \*Sender)  {  Form2 = new TForm2(this);  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::Quitter1Click(TObject \*Sender)  {  Close();  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::Cascade1Click(TObject \*Sender)  {  Cascade();  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::Horizontale1Click(TObject \*Sender)  {  TileMode = tbHorizontal;  Tile();  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::Verticale1Click(TObject \*Sender)  {  TileMode = tbVertical;  Tile();  }  Vous pouvez dès maintenant compiler l'application. Vous pouvez vous apercevoir qu'avec peu de code elle est déjà fonctionnelle.  Vous pouvez voir que les fiches enfants occupent toute la zone client de la fenêtre parent. On peut se réserver une partie de la fiche parent pour mettre par exemple des boutons comme sur la capture d'écran que vous avez pu voir au début de cet article. Pour cela nous poserons sur la fiche un TPanel avec les propriétés suivantes :    **Align** à alLeft    **Caption** à Vide     **BevelOuter** à bvNone  Sur ce panel vous pouvez poser ce dont vous avez besoin. Dans la capture d'écran en exemple j'avais posé 2 SpeedButtons dont j'avais affecté les événements OnClick aux méthodes des menus New1Click et Quitter1Click.  **Questions diverses :**  **Les menus :** Si vos fenêtres enfants possèdent un MainMenu, il prendra la place du MainMenu de la fenêtre parent dès que l'une d'elle est ouverte. Si vous voulez fusionner les menus, il faudra utiliser leurs propriétés **GroupeIndex** afin de les organiser.  **Accès à la fiche enfant active depuis la fiche principale :** Pour accéder à la fiche enfant qui détient le focus depuis la fiche principale on utilise sa propriété **ActiveMDIChild**. Par exemple nous allons ajouter dans notre menu "Fichier" l'option "Fermer" qui devra fermer la fiche active. Donc sur l'événement OnClick de "Fermer" rajouter ceci :  //...  void \_\_fastcall TForm1::Fermer1Click(TObject \*Sender)  {  if(ActiveMDIChild) ActiveMDIChild->Close();  }  //...  Vous remarquerez qu'avant de fermer une fiche enfant nous testons qu'il en existe bien une. (ActiveMDIChild a la valeur NULL si aucune fiche enfant n'est présente.)  **Accès à un objet de la fiche enfant active depuis la fiche principale :** Par exemple effacer le contenu de Memo1 de la fiche enfant active depuis la fiche principale. Créer une nouvelle option de menu avec son Caption à Clear, sur son événement OnClick mettre :  //...  void \_\_fastcall TForm1::Clear1Click(TObject \*Sender)  {  if(ActiveMDIChild)  ((TForm2 \*)ActiveMDIChild)->Memo1->Lines->Clear();  }  //...  Dans ce cas on est obligé d'effectuer un changement de type en pointeur sur TForm2 car le type de ActiveMDIChild est un pointeur sur TForm.  **Accès à toutes les fiches enfants :** La fiche parent peut aussi accéder à ses fiches enfants par un indice à l'aide de la propriété MDIChildren. Exemple pour fermer toutes les fiches enfants, vous créez une nouvelle option dans le menu "Fichier" : "Tout fermer" sur son événement OnClick mettre :  //...  void \_\_fastcall TForm1::Toutfermer1Click(TObject \*Sender)  {  for(int x = MDIChildCount-1; x>=0; x--)  MDIChildren[x]->Close();  }  //...  Attention toutefois à cet indice : il change en fonction de l'ordre auquel nous avons accédé aux fiches précédemment, la dernière fiche accédée a l'indice 0, donc évitez d'utiliser cet indice pour accéder à une fiche particulière.  **Passer le focus à la fiche suivante ou précédente :** Pour cela on utilise les méthodes de la fiche parent **Next** et **Previous**. Vous pouvez remarquer que le changement de focus ce fait encore en fonction de l'indice.   Vous avez maintenant quelques bases pour construire votre première application MDI. |

**TThread**

**Création d'un thread avec la classe TThread:**

Ce tutoriel va vous présenter sous forme d'un exemple, comment créer un thread avec la classe TThread de la VCL de C++Builder.   
  
Tout d'abord qu'est qu'un thread ?  
On peut voir ça comme une partie de code qui s'exécute en parallèle au code principal de notre application, qui en fait est aussi un thread. Le système d'exploitation (sous les systèmes monoprocesseur) "partage" le temps processeur entre les différents threads en donnant régulièrement accès à chacun d'eux au processeur pendant un très court instant, ce qui nous donne l'impression que les deux parties de code s'exécutent simultanément. Cela peut être utile pour exécuter des processus lents sans bloquer l'application.   
  
Pour aborder la classe TThread nous allons prendre un sujet qui a été traité plusieurs fois sur le forum C++ Builder: Comment lancer une application externe et attendre qu'elle se termine pour redonner la main à notre application ?   
  
Nous lancerons donc notre application externe avec le code suivant : (Pour l'exemple ce sera la calculatrice de Windows qui en principe se trouve sur toutes les configurations.)

SHELLEXECUTEINFO shInfo;

ZeroMemory(&shInfo,sizeof(shInfo));

shInfo.cbSize=sizeof(shInfo);

shInfo.hwnd=NULL;

shInfo.fMask=SEE\_MASK\_NOCLOSEPROCESS;

shInfo.lpVerb=NULL;

shInfo.lpFile="Calc.exe";

shInfo.lpParameters=NULL;

shInfo.nShow=SW\_SHOWNORMAL;

bool shRetour=ShellExecuteEx(&shInfo);

if (shRetour) WaitForSingleObject(shInfo.hProcess,INFINITE);

Dans ce code la méthode WaitForSingleObject bloque l'application tant que l'application appelée n'est pas terminée. Ce qui a pour effet néfaste que le dessin de notre application n'est plus rafraîchi. Il ne va pas sans dire que ce n'est pas très esthétique. Donc pour remédier à cet inconvénient, nous allons lancer l'application externe dans un thread et nous désactiverons les événements souris et clavier de notre application tant que le thread n'est pas terminé. Par contre le dessin de la fenêtre de notre application sera toujours opérant.   
  
Pour l'exemple créons une nouvelle application, puis créons le thread, pour cela faîtes "Nouveau" puis sur l'onglet "Nouveau" : "Objet Thread" , saisir le nom de sa classe dans la boîte de dialogue prévue à cet effet et faire "OK", pour l'exemple nous l'appellerons "TMonThread". Maintenant nous avons nos fichiers Unit2.h et Unit2.cpp contenant le code minimum de notre thread.   
Il ne reste plus qu'à ajouter le code pour lancer l'application externe dans la méthode **execute** du thread.   
  
Fichier Unit2.h :

#ifndef Unit2H

#define Unit2H

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

#include <Classes.hpp>

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

class TMonThread : public TThread

{

private:

protected:

void \_\_fastcall Execute();

void \_\_fastcall AffMessage();

public:

\_\_fastcall TMonThread(bool CreateSuspended);

};

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

#endif  
Fichier Unit2.cpp :

#include <vcl.h>

//#include <shellapi.h> // nécéssairesous BCB4

#pragma hdrstop

#include "Unit2.h"

#include "Unit1.h"

#pragma package(smart\_init)

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

// Important : les méthodes et les propriétés des objets de la VCL ne peuvent être

// utilisées que dans une méthode appelée en utilisant Synchronize, comme :

//

// Synchronize(UpdateCaption);

//

// où UpdateCaption serait de la forme :

//

// void \_\_fastcall TMonThread::UpdateCaption()

// {

// Form1->Caption = "Mise à jour dans un thread";

// }

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

\_\_fastcall TMonThread::TMonThread(bool CreateSuspended)

: TThread(CreateSuspended)

{

}

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

void \_\_fastcall TMonThread::Execute()

{

//---- Placer le code du thread ici ----

SHELLEXECUTEINFO shInfo;

ZeroMemory(&shInfo,sizeof(shInfo));

shInfo.cbSize=sizeof(shInfo);

shInfo.hwnd=NULL;

shInfo.fMask=SEE\_MASK\_NOCLOSEPROCESS;

shInfo.lpVerb=NULL;

shInfo.lpFile="Calc.exe";

shInfo.lpParameters=NULL;

shInfo.nShow=SW\_SHOWNORMAL;

bool shRetour=ShellExecuteEx(&shInfo);

if (shRetour)

{

Synchronize(AffMessage);

WaitForSingleObject(shInfo.hProcess,INFINITE);

}

}

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

void \_\_fastcall TMonThread::AffMessage()

{

Form1->Label1->Caption = "Application externe lancé.";

}

Cette fois la fonction WaitForSingleObject bloquera uniquement le Thread et pas l'application. Nous y ajouterons une méthode nommé AffMessage qui nous changera le texte d'un Label se trouvant sur la fiche principale de l'application. Ceci juste pour mettre en oeuvre ce qui est expliqué en commentaire dans le code généré de "Unit2.cpp". Cette méthode sera appellée par la méthode Synchronize dès le démarrage de l'application externe. Comme expliqué ce mode opératoire doit être appliqué dés que l'on utilise des méthode ou des propriété de la VCL.   
Maintenant il faut lancer notre Thread depuis notre fenêtre principale. Pour l'exemple nous poserons sur la fiche (Form1) un bouton (Button1) et un Label (Label1). Voyons maintenant son code.   
  
Fichier Unit1.h :

#define Unit1H

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

#include <Classes.hpp>

#include <Controls.hpp>

#include <StdCtrls.hpp>

#include <Forms.hpp>

#include "Unit2.h"

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

class TForm1 : public TForm

{

\_\_published: // Composants gérés par l'EDI

TButton \*Button1;

void \_\_fastcall Button1Click(TObject \*Sender);

void \_\_fastcall FormClose(TObject \*Sender, TCloseAction &Action);

private: // Déclarations de l'utilisateur

TMonThread \*MonThread;

void \_\_fastcall FinDeMonThread(TObject \*Sender);

public: // Déclarations de l'utilisateur

\_\_fastcall TForm1(TComponent\* Owner);

};

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

extern PACKAGE TForm1 \*Form1;

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

#endif

Dans le fichier Unit1.h nous incluons le fichier entête "Unit2.h" pour avoir accès à la classe TMonThread, nous déclarons un pointeur MonThread sur un objet TMonThread et nous déclarons une méthode FinDeMonThread qui sera appelée quand le thread sera terminé.   
  
Fichier Unit1.cpp :

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"

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

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

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

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner)

{

}

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

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

MonThread = new TMonThread(false);

MonThread->OnTerminate = FinDeMonThread;

EnableWindow(Handle, false);

}

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

void \_\_fastcall TForm1::FinDeMonThread(TObject \*Sender)

{

EnableWindow(Handle, true);

Label1->Caption = "Thread terminé";

delete MonThread;

MonThread = NULL;

}

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

void \_\_fastcall TForm1::FormClose(TObject \*Sender, TCloseAction &Action)

{

if(MonThread) Action = caNone;

}

Sur l'événement OnClick de Button1, nous créons le Thread MonThread avec l'opérateur new, puis nous affectons la méthode FinDeMonThread à l'événement OnTerminate de MonThread, événement qui est déclanché comme son nom l'indique quand le thread est terminé et puisque nous voulons que la fiche Form1 ne reçoive plus les événements clavier et souris pendant l'exécution du thread nous les désactivons avec la fonction de l'API Windows EnableWindow. Quand le Thread est terminé, la méthode FinDeMonThread est exécutée, nous réactivons le clavier et la souris pour la fiche principale et nous y détruisons le Thread. Sur l'événement OnClose de Form1 (méthode FormClose) nous nous s'autorisons à quitter l'application seulement si le thread est terminé, car si le clavier et la souris n'est plus actif pour l'application, le clique droit sur le bouton de l'application dans la barre des taches et lui encore accéssible et permet de fermer l'application.   
  
Ce petit exemple montre comment créer un Thread avec C++ Builder, comme vous pouvez le voir la classe TThread de la VCL nous facilite encore grandement les choses.   
  
Ce code à été élaboré sous BCB6. Je l'ai testé sous BCB4, pour que la compilation aboutisse il a fallu inclure le fichier entête "shellapi.h" dans "Unit2.cpp".

**Installation et utilisation de wxWidgets avec C++ Builder**

**Introduction :**  
Dans cet article nous allons aborder l'installation de wxWidgets et son utilisation dans l'EDI de C++ Builder 6.   
Tout d'abord qu'est ce que wxWidgets?  
**wxWidgets** anciennement nommé wxWindows est une bibliothèque de classes destinée à la conception d'applications fenêtrées, telle que les applications Windows GUI. L'avantage de cette bibliothèque est qu'elle permet d'écrire des applications portables sur différentes plateformes.(Windows, Linux, Unix, MacOs...)  
Installée dans l'EDI de C++Builder elle sera plus facile d'utilisation, projets préconfigurés, achèvement du code opérationnel.

**Installation de la bibliothèque:**

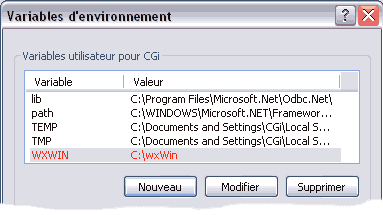
Après avoir téléchargé l'archive sur le site de [wxWidgets](http://www.wxWidgets.org/) décompressez-là dans un dossier temporaire, puis exécutez le programme d'installation "setup.exe". Il n'y a pas grand chose à dire sur le déroulement de l'installation, personnellement j'ai juste modifié le nom du dossier d'installation avec un nom plus court "C:/wxWin". Nous utiliserons ce nom pour la suite de ce document.

**Compilation de la bibliothèque :**

Avant de lancer la compilation il faut définir une variable d'environnement nommée WXWIN et lui donner comme valeur le dossier où vous avez installé wxWidgets.  
**Sous windows 9x** la définir dans l'autoexec.bat :

**set WXWIN=c:\wxWin**

**Sous Windows XP** ouvrez le panneau de configuration puis l'applet "Système", sur l'onglet avancé cliquez sur le bouton "Variable d'environnement" et ajoutez-là.  
(En rouge sur la copie d'écran ci-dessous)

  
Cette opération terminée rendez-vous dans le dossier "C:\wxWin\src\msw" où vous trouverez les fichiers sources de la bibliothèque, qu'il va falloir compiler.  
  
**Compilation des source en ligne de commande :**  
La compilation s'effectuera en ligne de commande dans une console ou à l'aide de fichiers batchs ".bat" en utilisant l'utilitaire Make.exe inclut avec C++ Builder qui sert à exécuter des fichiers Make. Ceux destinés à être compilés avec C++ Builder ont l'extension ".B32" et sont inclus dans le package.

**make -f makefile.b32**

La compilation peut paraître un peu longue, soyez patient...  
Une fois terminée elle aura produit une librairie statique nommée "wx24ds\_bcc.lib" placée dans le dossier "C:\wxWin\lib".   
  
**Compilation d'un exemple en ligne de commande :**  
La compilation terminée rendez-vous dans le dossier "C:\wxWin\samples\minimal", nous allons y compiler l'exemple en ligne de commande pour tester si tout fonctionne:

**make -f makefile.b32**

Même commande que pour la compilation des sources car chaque exemple a ses propres fichiers Makes. Si la compilation s'est bien déroulée elle doit avoir produit un fichier nommé "minimal.exe".  
  
**Compilation des différentes versions de la librairie:**  
La précédente compilation des sources avait produit la librairie Debug. Pour obtenir la librairie Release rajoutez 'FINAL=1' comme argument :

**make -f makefile.b32 clean**

**make -f makefile.b32 FINAL=1**

Produit le fichier "wx24s\_bcc.lib" toujours dans le dossier "C:\wxWin\lib\". La 1ère ligne avec l'argument clean sert à nettoyer les précédentes compilations.   
Et pour produire la librairie sous forme de dll rajoutez 'DLL=1' comme argument:

**make -f makefile.b32 clean**

**make -f makefile.b32 DLL=1 FINAL=1**

Production ici de la dll: wx24\_bcc.dll que vous pouvez mettre dans le dossier système de windows et de sa librairie d'importation: "wx24\_bcc.lib".  
Il y a d'autres possibilités de compilations, dans ce document je me suis limité à ces trois. (Voir le fichier "install.txt" situé dans le dossier "C:\wxWin\docs\msw")   
  
**Problèmes rencontrés :**  
La commande Make avec l'argument clean m'a causé un souci lors de mes premiers essais. Si vous l'exécutez après la dernière compilation elle efface les librairies annexes du dossier "C:\wxWin\lib" comme j'ai laissé les librairies dans ce dossier le linkeur ne les trouvait plus.

**Utilisation dans l'EDI de C++ Builder :**

Pour l'utiliser sans trop de contraintes dans C++ Builder, j'ai modifié le fichier header de pré compilation "C:\wxWin\include\wx\wxprec.h" où j'ai rajouté les lignes en bleu dans le code ci-dessous juste après la directive '#include "wx/defs.h"' ceci afin que le linkeur sélectionne automatiquement la bonne librairie. Ceci évite d'avoir à les rajouter ou les changer si l'on change le type de compilation. (debug release ou dll) Les noms des librairies peuvent être différents selon la version de la librairie. (J'ai limité la modification aux trois compilations précédemment effectué.)

// compiler detection; includes setup.h

#include "wx/defs.h"

// Ajouté pour C++ Builder

#ifdef \_\_BORLANDC\_\_

#if !defined(WXUSINGDLL)

#if defined(\_DEBUG)

#pragma link "wx24ds\_bcc.lib" // debug

#else

#pragma link "wx24s\_bcc.lib" // release

#endif

#else

#if !defined(\_DEBUG)

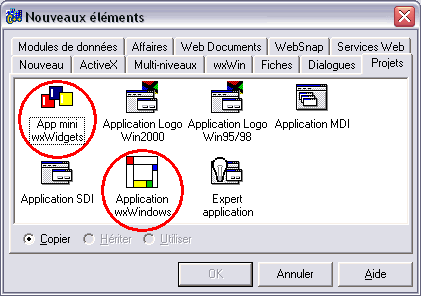
#pragma link "wx24\_bcc.lib" //dll release

#endif

#endif

#endif

Pour une utilisation encore plus aisée nous allons créer un modèle de projet que vous pourrez utiliser ultérieurement pour créer des projets vides prêts à l'emploi. Pour arriver à nos fins nous utiliserons le référentiel d'objets. (Voir capture écran ci-dessous)



Tout d'abord créer un dossier nommé "wxWidgets" ou d'un nom de votre choix dans le dossier "ObjRepos" de C++ Builder et placer le fichier "mondrian.ico" que vous trouverez dans wxWidgets dans ce dossier. Créer un nouveau projet par l'intermédiaire de l'expert Console.(Menu: Fichier -> Nouveau -> Autres... -> Onglet Nouveau -> Expert Console) Dans la boîte de dialogue "Expert Console" seul doit rester coché le bouton radio "C++" faire 'Ok'.  
Effacer le contenu du fichier source ".cpp" et remplacer par :

#include "wx/wxprec.h"

#ifdef \_\_BORLANDC\_\_

#pragma hdrstop

#endif

#ifndef WX\_PRECOMP

#include "wx/wx.h"

#endif

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

#include "wxWin.h"

IMPLEMENT\_APP(MyApp)

**bool**  MyApp::OnInit()

{

MyFrame \*frame = **new** MyFrame(\_T("Minimal wxWindows App"),

wxPoint(100, 100), wxSize(480, 360));

frame->Show(TRUE);

**return** TRUE;

}

MyFrame::MyFrame(**const** wxString& title, **const** wxPoint& pos, **const** wxSize& size,

**long** style) : wxFrame(NULL, -1, title, pos, size, style)

{

}

Faire enregistrer, dans le dossier wxWidgets préalablement créé, changer le nom du fichier en "wxWin.cpp"  
  
Puis créer un fichier header avec :

#ifndef wxWinH

#define wxWinH

**class** MyApp : **public** wxApp

{

**public:**

**virtual bool** OnInit();

};

**class** MyFrame **: public** wxFrame

{

**public:**

MyFrame(**const** wxString& title, **const** wxPoint& pos, **const** wxSize& size,

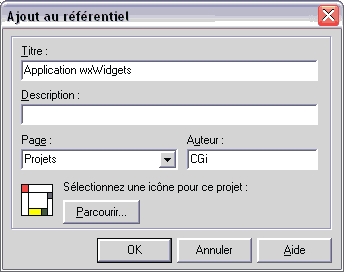
**long** style = wxDEFAULT\_FRAME\_STYLE);

**private:**

};

#endif //wxWinH

Enregistrer toujours dans le même dossier sous le nom "wxWin.h".  
Nous allons maintenant éditer les options de projet.(Menu: Projet -> Options...)  
**Onglet :**  
**Application :** sélectionner l'icône "mondrian.ico"  
**Compilateur :** Cocher 'Traiter enum comme int (z)' (Impératif) et cocher l'avertissement W8057 (bouton 'Avertissements')  
**Options avancées du compilateur :** Mettre Alignement des données à Byte. (Impératif)  
**C++ :** Décocher 'Classe de base vide long. zéro' et 'Membres classes vide long. zéro'  
**Répertoire/Conditions :**Dans chemin d'inclusion ajouter le dossier "include" de wxWidgets "C:\wxWin\include" (Impératif), dans chemin de bibliothèque ajouter le dossier lib de wxWidgets "C:\wxWin\lib" (Impératif)  
Pour les définitions conditionnelles quelques sites Internet en donnent une liste à définir, après quelques tests que j'ai effectués la plupart se trouvaient être déjà définies sauf \_\_WXWIN\_\_ et USE\_DEFINE dont je n'ai trouvé trace et la compilation s'effectue sans problèmes même si elles ne sont pas définies. A voir au cas par cas, il est probable que selon les versions de wxWidgets et C++ Builder il y ait des différences.  
La constante \_DEBUG est automatiquement ajoutée selon que vous sélectionnez le mode Débogage ou non (version finale) dans le projet.   
  
Fermer la boîte de dialogue Projet par 'Ok' puis sauvegarder le projet toujours dans le même dossier sous le nom "wxWinProjet.bpr" (le nom du projet sans l'extension ne doit pas être le même que celui du fichier source sans l'extension.)  
Tout en gardant le projet ouvert nous allons l'ajouter au référentiel pour qu'il apparaisse dans la boîte de dialogue Nouveau Projet comme la copie d'écran au début de ce paragraphe. Pour cela faire Menu: Projet -> Ajouter au référentiel, remplir la boîte de dialogue comme la copie d'écran ci-dessous sélectionner l'icône que l'on avait mis dans le dossier et faire Ok.



Fermer le projet.  
Maintenant créons un nouveau projet selon le modèle préalablement défini. Menu : Fichier -> Nouveau -> Autres... Onglet Projet sélectionner l'icône Application wxWidgets, ceci ouvre une boîte de sélection de répertoire. Positionnez-vous dans le dossier où vous voulez créer votre application et faites Ok.  
Vous pouvez déjà compiler votre projet vide il doit vous créer une fenêtre Windows. Si vous voulez le compiler en utilisant la dll il faudra définir la constante de compilation conditionnelle 'WXUSINGDLL=1'

**Programmez avec wxWidgets :**

Pour faire vos premiers pas avec la bibliothèque, vous pouvez consulter cette série d'articles : [Initiation à wxWidgets.](http://chgi.developpez.com/wxwidgets/)

**Remarques sur ce document :**

Ce document relate l'installation de wxWidgets telle que je l'ai vécue. C'est-à-dire celle de la version 2.4.2 avec C++ Builder 6 sous Windows XP. Il est possible qu'il y ait des différences sous d'autres configurations.  
Vous pouvez vous référer à la documentation incluse avec le package notamment le fichier install.txt dans le dossier "C:\wxWin\docs\msw" pour plus de précisions sur l'installation. Pour l'utilisation de la bibliothèque des fichiers d'aide très complets (malheureusement en anglais) ainsi que de nombreux exemples accompagnent le package.   
  
Bonne installation,

**http://chgi.developpez.com/wxwidgets/**