Chapitre 11

Les interfaces graphiques

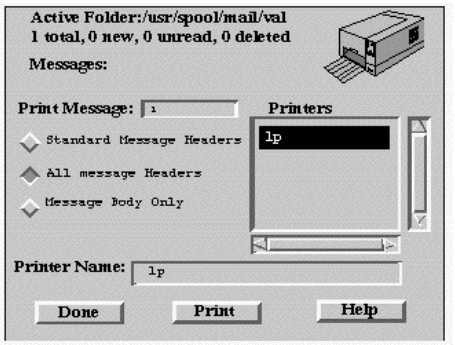
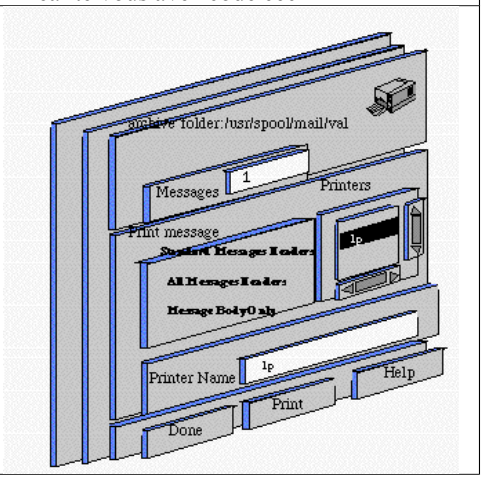
(wxWidgets)

1. Définition de l'interface graphique

- Une interface graphique est un ensemble de programmes permettant d'utiliser un système donné de manière plus souple et intuitive que les lignes de commande qui sont parfois difficiles à comprendre.

- Avec la rapidité croissante de l'internet, de plus en plus des personnes ou des sociétés aimeraient offrir des services ou contrôler divers équipements à partir du réseau.

- L'interface graphique côté utilisateur devra donc fonctionner indépendamment du matériel, et doit agir comme un simple client recevant et envoyant des données critiques (nécessitant des mesures de sécurité appropriées) ou des données non critiques.

Quand vous voyez ceci	En réalité vous avez codé ceci
	

- Il existe donc des composants graphiques qui en contiennent d'autres (et gèrent leur apparition, leur positionnement ...) et d'autres qui n'en contiennent pas tels les boutons poussoirs, les fenêtres textes de saisie ... Ces derniers sont parfois appelés les contrôles.

2. Boîtes à outils et Langages de programmation

- **Java**
  - Développé par la compagnie SUN en 1995.
  - Indépendant de la plateforme.
  - AWT est le paquetage de base pour construire et manipuler des interfaces graphiques. Il est parmi les paquetages originaux de Java.
  - Swing est le nouveau paquetage. Ses composantes sont écrites, manipulées et affichées complètement en Java ("pur" java).
  - Java est dans sa version 8.
- **Microsoft**
  - Avant 2001, il y avait deux possibilités : Visual Basic ou bien Visual C/C++ (MFC).
    - Visual Basic : pour des applications simples, pas complexes de tous les jours.
    - Visual C/C++ (MFC : Microsoft Foundation Classes) : pour les interfaces plus complexes nécessitant une forte interaction avec le gestionnaire de fenêtre à la Windows, etc.

- L'interface de développement était « Visual Studio 6 ».
- .NET (depuis 2000)
    - Sa naissance juillet 2000, et c'est le premier « framework » dont la version 1.0 a été rendue publique le 15 janvier 2002.
    - L'aspect le plus intéressant de « .NET » se situe au niveau de la plateforme de développement et des langages qu'il met en avant.
    - Il a permis d'unifier l'environnement de développement.
    - Avant son arrivée, le choix d'un outil ou d'une technologie de développement de Microsoft était une tâche assez ardue! L'explication est que les solutions étaient vastes et pouvaient impliquer un des gadgets Visual Basic 6.0, que C++, VBScript, MFC, DCOM, ATL, etc.
    - Par ailleurs, Microsoft perdait du terrain devant la concurrence de SUN et son langage Java.
    - Il fallait une solution intégrée, ouverte vers le web.
    - L'objectif est donc le développement de manière simple d'applications web inter portables d'où l'arrivée de « .NET ».
    - En date de mars 2015, la version actuelle du « framework » est 4.5.2, alors que celle de l'interface de développement est « Visual Studio 2013 ».

- **C/C++ :**

Qt : développée par « Trolltech ».
Langage : C++.
Disponible sous licence GPL ou commerciale.
Disponible sous GPL depuis peu à la communauté Windows.
Exemple : KDE, gestionnaire de fenêtre sous Linux. Navigateur Opera.

wxWidgets : développée par la communauté « Open Source ».
Langage : C++.
L'ancien nom « wxWindows ».
Disponible sous licence LGPL.
Disponible depuis plus d'une dizaine d'années.
Exemple : AOL Communicator.

GTK+/GTKMM : développée par la communauté « Open Source ».
Langage : C (gtkmm C++)
Elle a vu le jour à cause des contraintes de licences associées à l'utilisation de Qt.
Version Windows chaotique ! Il faut être sacrément très patient !
Exemple : Gnome, gestionnaire de fenêtre sous Linux. Gimp (images).

- Pour la suite de ce cours, nous allons décrire la bibliothèque « **wxWidgets** ».

3. Boîte à outils « wxWidgets »

3.1. Généralités

- « wxWidgets » est un logiciel libre qui est composé d'un ensemble d'outils, regroupés dans une bibliothèque et conçus pour fonctionner les uns avec les autres de façon complémentaire.
- « wxWidgets » est sous License LGPL (Library General Public Licence). Ceci facilite la redistribution des binaires.
- « wxWidgets » a l'avantage de fonctionner avec la plupart des compilateurs.
- « wxWidgets » donne à l'interface graphique l'apparence du système d'exploitation sur laquelle elle a été développée. Ceci permet d'éviter le dépaysement.
- « wxWidgets » est disponible sur une multitude de plateformes Windows, Linux, Unix, MacOS, Palm, etc.
- Cette boîte à outils a vu le jour en 1992 suite au travail de « Julian Smart », un chercheur de l'université d'Edinburgh.
- Julian Smart a développé la librairie pour réaliser des interfaces graphiques sous Windows (w) et Sun (x) et ceci, dans un environnement fenêtré (Windows Manager) d'où le nom de départ « wxWindows ».
- Au départ, cette librairie visait une compatibilité avec MFC 1.0 (Windows) et XView (Sun).

- Des modifications ont été apportées à la librairie pour supporter un mode natif win32. Elle a laissé tomber XView au profit de Xt et Motif.
- Par la suite, la librairie « wxWidgets » a été complètement repensée pour la rendre claire, efficace et portable à d'autres systèmes. Ainsi, en 1997 la version 2 de l'API a vu le jour.
- Depuis des classes non graphiques ont été ajoutées pour supporter diverses opérations comme la programmation réseau, le multitâche, etc.
- En 2004, à la demande Microsoft, « wxWindows » a été renommée « wxWidgets ».

3.2. Outils nécessaires pour utiliser « wxWidgets »

- La librairie peut-être téléchargée à partir de cette adresse : http://www.wxwidgets.org/
- En réalité, vous n'allez télécharger que le code source de cette librairie.
- Il faudra donc la compiler par un compilateur donné pour une plateforme donnée. Par la suite, il faudra l'installer pour qu'elle soit disponible durant le processus de compilation d'un programme à base de « wxWidgets ».
- Toutes les informations relatives à une installation et une utilisation sous Windows sont regroupées dans des fichiers ou répertoires dont le nom comporte le tag « msw ». À voir par exemple le répertoire « CHEMIN_wxWidgets\docs\msw ».

- Pour générer cette librairie, nous vous conseillons d'exécuter d'abord la commande « configure » avant celle de « make ». En effet, la commande « configure » permet d'ajuster les paramètres du « makefile » utilisé avec la commande « make » pour tenir compte de votre environnement de travail.

- Par le passé, nous utilisions dans le cours un logiciel qui intègre de manière transparente la librairie « wxWidgets ». Il s'agit du logiciel « wx-devcpp » disponible à partir de cette adresse : http://wxdsgn.sourceforge.net.

- Malheureusement, les versions du compilateur « g++ » et de la librairie « wxWidgets » utilisées par « wx-devcpp » n'étaient plus maintenues à jour. De ce fait, depuis la session 2014, le logiciel « codelite » a pris le relais. Le guide d'installation disponible sur le site web du cours explique comment installer ce logiciel pour Linux, Mac et Windows.

3.3. Quelques Tutoriaux

- Il y a deux manières de décrire l'utilisation de « wxWidgets » :

  - Soit utiliser un outil intégré pour réaliser des interfaces graphiques.
  - Soit coder « manuellement » ces interfaces graphiques. Voir pour cela le tutoriel de « David BEECH » :
    - (FR) http://phenix.developpez.com/Tutoriel_wxWindows/Tutoriel%20WxWindows_1.htm
    - (EN) https://wiki.wxwidgets.org/Guides_%26_Tutorials

Pour ce cours, nous allons suivre l'approche « manuelle » et nous allons de temps à autre développer les exemples du tutoriel de « David BEECH ».

3.4. Livres de référence

Cross-Platform GUI Programming with wxWidgets
Stefan Csomor, Kevin Hock, Julian Smart.
Prentice Hall.

- Une copie électronique est disponible à partir de cette adresse :

http://www.phptr.com/content/images/0131473816/downloads/0131473816_book.pdf

- Le « wiki » de « wxWidgets » :

https://wiki.wxwidgets.org/Main_Page

4. Développement d'une interface graphique à l'aide de « wxWidgets »

Une interface graphique avec « wxWidgets » consiste en :

- Une application objet ; chaque application « wxWidgets » doit dériver de la classe « wxApp ».
- Un objet cadre « frame » ; c'est la fenêtre principale. Elle contient une bordure et un titre. Un cadre peut avoir des objets comme une barre de menu, une barre d'état, une barre d'outils, une icône, etc. Il ne faudra pas confondre cette fenêtre avec la fenêtre de premier niveau (« Window ») qui est sans bordure (aucun affichage à l'écran). La « frame » est une instance de la classe « wxFrame ». Cette classe fournit une fenêtre dont la taille et la position peuvent être modifiées par le programmeur.
- Les deux classes « wxApp » et « wxFrame » font partie de « wxWindows ». Pour pouvoir les utiliser il faudra inclure dans le programme le fichier d'en-tête « wx/wx.h ».

4.1. Création d'une application « wxWidgets »

- Pour créer une application « wxWidgets », on élabore d'abord une classe qui dérive de la classe « wxApp ».
- L'application va être représentée par une seule instance de cette classe dérivée.
- Dans la classe dérivée, on surcharge la méthode « bool OnInit( ) ».

- Au lancement de notre application, le programme va exécuter automatiquement la méthode virtuelle « bool OnInit( ) ». Par défaut cette méthode ne fait rien. Nous allons la redéfinir en conséquence pour lui demander d'ouvrir la fenêtre principale de notre application.
- La méthode « bool OnInit() » renvoie « true » si les opérations se sont déroulées correctement, elle renvoie « false » dans le cas contraire. Dans ce cas, l'application est détruite.
- Dans la méthode « OnInit », nous allons créer l'objet-cadre. Nous allons définir sa forme et son comportement.

```
// Fichier « EditeurTxTApp.h »

#ifndef EDITEURTXTAPP_H
#define EDITEURTXTAPP_H

// Déclare la classe Application « EditeurTxTApp »
// La déclaration minimale que l'on puisse écrire

class EditeurTxTApp : public wxApp {
public:
       // Cette méthode doit être redéfinie.
       // Elle sera appelée au démarrage de l'application
       virtual bool OnInit();
};

#endif
```

```
// Fichier « EditeurTxTApp.cpp »

#include <wx/wx.h>
#include " EditeurTxTApp.h"


// Le code pour créer une instance de « EditeurTxTApp » est
// interne, caché. Vous devez quand même dire à « wxWidgets »
// la nature de l'objet à créer. Pour cela, nous allons
// faire appel à la macro « IMPLEMENT_APP ». Cette
// dernière va demander à « wxWidgets » de créer l'objet
// passé en argument ici « EditeurTxTApp ».
// La macro remplace en quelque sorte le point d'entrée i.e.
// la fonction « main », ou « int WINAPI WinMain( ... ) »
// la fonction associée à une interface graphique

IMPLEMENT_APP(EditeurTxTApp)

// Code de l'initialisation de l'application

bool EditeurTxTApp::OnInit() {

  // On crée une instance de la classe « wxFrame »
  // On définit le texte qui s'affichera dans le haut
  // de la fenêtre, son emplacement et sa taille.

  wxFrame *frame =
                  new wxFrame((wxFrame*) NULL, -1,
                          "Editeur de texte simpliste");
```



```
  // On rend visible cette « frame ». Cette méthode ne fait
  // pas partie de la classe « wxFrame », mais plutôt de
  // « wxWindow ». « wxFrame » a accès à cette méthode, car
  // cette classe dérive de « wxWindow ». Il faut avoir
  // toujours en tête une idée du contenu des classes
  // supérieures

  frame->Show(TRUE);

  // On informe que notre « frame » est la fenêtre principale.
  // Ainsi l'appel à cette méthode permet de mettre la
  // « frame » au-dessus de toutes les fenêtres afin d'avoir
  // le focus.

  SetTopWindow(frame);

  // Retourne « true » si la création s'est bien déroulée
  return true;

}
```

- Nous n'avons mentionné nulle part dans le fichier « `EditeurTxTApp`.cpp » un appel pour détruire la « frame », une instance de « `EditeurTxTApp` ». La raison est toute simple : en mettant la « frame » comme étant la fenêtre principale de notre application, l'application va se charger de la détruire pour nous lors de la fermeture de cette dernière.

- Le résultat obtenu est comme suit :

4.2. Utilisation de « wxFrame »

- La classe « wxFrame » nous fournit les éléments d'une fenêtre standard, une fenêtre, dont la taille peut-être modifiée.
- Comme dans l'exemple précédent, une fenêtre va avoir une bordure épaisse et un titre.
- Une fenêtre peut avoir aussi optionnellement une barre de menu, une barre d'état, une barre d'outils, une icône, etc.
- Une « frame » peut agir comme conteneur d'autres composantes (boutons, zone de texte, etc.), mais pas une autre fenêtre ni une autre « frame ».
- « wxFrame » dérive de « wxWindows ».

4.2.1 Création d'une « frame »

- Normalement, nous devons concevoir une classe qui dérive de la classe « wxFrame ».
- Cette manière va nous permettre d'ajouter des fonctionnalités propres à notre classe, tout en profitant des fonctionnalités de la classe de base « wxFrame ».

```
// Fichier « EditeurTxTApp.h »

#ifndef EDITEURTXTAPP_H
#define EDITEURTXTAPP_H

class EditeurTxTApp : public wxApp {
public:
       virtual bool OnInit();
};

// Création de la classe associée à la fenêtre principale
// qui va contenir un titre, une bordure, etc.
class TexteFrame : public wxFrame {
public:
       // constructeur
       TexteFrame( const wxString titre, int xpos, int ypos,
                   int width, int height);
       // destructeur
       ~TexteFrame();
};
#endif
```

```
// Fichier " EditeurTxTApp.cpp "

#include <wx/wx.h>
#include "EditeurTxTApp.h"



IMPLEMENT_APP(EditeurTxTApp)


bool EditeurTxTApp::OnInit() {

  // Créer la fenêtre principale.
  // Elle a un nom, un point de départ (x,y),
  // une largeur et une hauteur

  const wxString Titre("Editeur de texte simpliste");

  TexteFrame *frame = new TexteFrame(Titre,50, 50, 450, 300);


  frame->Show(TRUE);

  SetTopWindow(frame);

  return true;

}
```



```
// Rien à ajouter de particulier dans le constructeur.
// On passe directement les paramètres au constructeur
// de la classe de base

TexteFrame::TexteFrame
             (const wxString titre,
                    int xpos, int ypos,
                    int width, int height)
             : wxFrame
                ( (wxFrame *) NULL,
                   -1,
                   titre,
                   wxPoint(xpos, ypos),
                   wxSize(width, height)
                )
{}

// Idem rien de spécial dans le destructeur

TexteFrame::~TexteFrame()
{}
```

- Le constructeur de « wxFrame » est appelé à partir du constructeur de « TexteFrame ».

- Le « cast » de « NULL » dans « `(wxFrame *) NULL` » est nécessaire pour assurer la portabilité entre les compilateurs. En effet certains compilateurs définissent « NULL » avec la valeur « 0L ». Dans ce cas aucune conversion vers un pointeur n'est permise. Pour toute la suite de ce cours, les pointeurs « NULL » seront forcement « casté » à un type approprié.

- Le constructeur de « wxFrame » est défini ainsi :

```
wxFrame(wxWindow* parent,
        wxWindowID id,
        const wxString& titre,
        const wxPoint& pos = wxDefaultPosition,
        const wxSize& dimension = wxDefaultSize,
        long style = wxDEFAULT_FRAME_STYLE,
        const wxString& nom = "frame");
```

- « parent » est un pointeur à la fenêtre parent. Dans l'exemple « `EditeurTxTApp` », la « frame » n'a aucun parent ce qui explique la valeur « `(wxFrame *) NULL` » passée comme premier argument.

- « id » est l'identificateur de la fenêtre.

- « titre » est le titre de la fenêtre. Il sera affiché en haut de la « frame ».

- « pos » c'est la position de la fenêtre. Il s'agit du coin en haut à gauche de notre fenêtre. La valeur est une instance de la classe « wxPoint ». Par exemple : pour positionner la « frame » aux coordonnées (5,20), il faut écrire « wxPoint(5,20) » au niveau de ce paramètre. Une valeur (-1,-1) qui correspond à la valeur par défaut « wxDefaultPosition », indique la position par défaut choisie par le gestionnaire de fenêtres.

- « dimension » correspond à la dimension de la fenêtre. La valeur est une instance de la classe « wxSize ». Par exemple : pour que la « frame » ait une taille de (100x200) pixels, il faut écrire « wxSize(200,400) » au niveau de ce paramètre. Une valeur (-1,-1) qui correspond à la valeur par défaut « wxDefaultSize », indique la taille par défaut choisie par le gestionnaire de fenêtres.

- « style » est le style de la fenêtre. « wxWidgets » définit plusieurs styles (voir la documentation) par exemple « wxDEFAULT_FRAME_STYLE ». Ce dernier définit la boîte par défaut qui contient les éléments suivants (wxMINIMIZE | wxMAXIMIZE_BOX | wxRESIZE_BOX | wxSYSTEM_MENU | wxCAPTION).

- « nom » est le nom de la « frame ». On l'utilise assez souvent pour associer la « frame » à un item.

```
const wxString Titre("Editeur de texte simpliste");

TexteFrame *frame = new TexteFrame(Titre,50, 50, 450, 300);

TexteFrame::TexteFrame
             (const wxString titre,
                    int xpos, int ypos,
                    int width, int height)
             : wxFrame
                ( (wxFrame *) NULL,
                   -1,
                   titre,
                   wxPoint(xpos, ypos),
                   wxSize(width, height)
                ){}
```

« titre » = « `Editeur de texte simpliste` », `wxPoint(50,50), wxSize(450, 300).`

- La classe « wxFrame » contient plusieurs méthodes membres comme « CreateStatusBar », « CreateToolBar », « GetTitle », etc.

4.2.2 Utilisation des contrôles

- Après avoir construit une application minimaliste, nous allons l'enrichir avec les outils nécessaires pour le traitement de texte.

- « wxTextCtrl » est la classe responsable de l'affichage et l'édition du texte.

- Il fournit les fonctionnalités d'un éditeur de texte comme : l'insertion, la sélection et la suppression de texte ; le « couper, copier, coller » ; le mouvement du curseur, etc.

- Le texte affiché peut-être sur une seule ligne ou plusieurs lignes c'est juste une question de « style ». Ce style est choisi au départ et permet d'indiquer à la fenêtre comment disposer le texte fourni à l'écran.

```
class TexteFrame : public wxFrame {
public:
       // constructeur
       TexteFrame( const wxString titre, int xpos, int ypos,
                   int width, int height);
       // destructeur
       ~TexteFrame();

private:
        // « letexte » est un pointeur vers un contrôleur
        // de texte « wxTextCtrl »
        wxTextCtrl *leTexte;
};
```


- On n'apporte des modifications que dans le constructeur de la classe comme suit :

```
TexteFrame::TexteFrame
             (const wxString titre, int xpos, int ypos,
                    int width, int height)
             : wxFrame
                ( (wxFrame *) NULL,
                   -1, titre, wxPoint(xpos, ypos),
                   wxSize(width, height)
                )
{

 // Initialisation du pointeur à la valeur "NULL castée"
 leTexte = (wxTextCtrl *) NULL;

 // "leTexte = new wxTextCtrl( ... )" contient un paramètre
 // "this" qui est une référence à la fenêtre parent.
 // Dans ce cas "this" pointe vers le cadre parent qui est
 // en train ou qui vient juste d'être construit.

 leTexte = new wxTextCtrl
              ( this,
                  -1,
                  wxString("Maintenant c'est à vous de taper
                              un texte ...\n\n"),
                  wxDefaultPosition,
                  wxDefaultSize,
                  wxTE_MULTILINE|wxTE_RICH2
              );
```

```
     // On fixe la couleur des caractères en rouge

      leTexte->SetDefaultStyle(wxTextAttr(wxColour(255,0,0)));
      leTexte->AppendText("En rouge?\n");

      // On fixe la couleur de fond en gris clair

      leTexte->SetDefaultStyle(wxTextAttr
                               (wxNullColour, *wxLIGHT_GREY));

      leTexte->AppendText("Sur fond gris?\n");

      // On change la couleur des caractères. Le fond reste le
      // même, c.-à-d. gris clair

      leTexte->SetDefaultStyle(wxTextAttr(*wxBLUE));
      leTexte->AppendText("En bleu sur fond gris?\n");


     }
```

- Le texte « Maintenant c'est à vous de taper un texte … » « En rouge ? » « Sur fond gris ? » « En bleu sur fond gris » sera affiché comme texte par défaut.

- Le parent de « wxTextCtrl » est « TexteFrame », on lui passe donc le pointeur « this ».

- Le constructeur de « wxTextCtrl » est défini comme suit :

```
wxTextCtrl(wxWindow* parent,
           wxWindowID id,
           const wxString& valeur = "",
           const wxPoint& pos = wxDefaultPosition,
           const wxSize& dimension = wxDefaultSize,
           long style = 0,
           const wxValidator& validator = wxDefaultValidator,
           const wxString& nom = wxTextCtrlNameStr);
```

- On remarque que le constructeur « wxTextCtrl » est similaire dans sa forme au constructeur de la classe « wxFrame ». En effet, les constructeurs des classes qui dérivent de « wxWindow » suivent le même pattern.

- « parent » est un pointeur à la fenêtre parent. Il ne doit pas être « NULL ».

- « id » est l'identificateur de la fenêtre. C'est un « id » pour permettre le contrôle de cet élément. « -1 » indique la valeur par défaut.

- « valeur » est le texte par défaut.

- « pos » c'est la position (x, y) du coin en haut à gauche.

- « dimension » correspond à dimension du contrôleur (largeur, hauteur).

- « style » est le style de la fenêtre. « wxTextCtrl » définit des styles additionnels. Nous avons pris en exemple « wxTE_MULTILINE » pour signifier que le contrôleur de texte permet d'afficher le texte sur plusieurs lignes et « wxTE_RICH2 », et ce, afin de profiter d'un contrôleur riche en fonctionnalités. Ce dernier sous le système « Windows » permet d'ajuster correctement la couleur des caractères, du fond de l'écran, etc. Ce paramètre est ignoré sur un système autre que « Windows ». Les styles peuvent être combinés en utilisant l'opérateur binaire « | » comme : « `wxTE_MULTILINE|wxTE_RICH2` » ou bien « `wxTE_MULTILINE|wxTE_RICH2|wxTE_CENTRE` » (plusieurs lignes avec un contrôleur riche de niveau 2, et le texte est justifié au centre).

- « validator » permet de valider les données qui sont transmises depuis le(s) contrôle(s) vers la structure de données du programme. Il peut être utilisé pour limiter les entrées exclusivement au format texte ou bien à des données numériques.

- « nom » est le nom de la fenêtre utilisée pour le contrôle.

- Là aussi, nous n'avons mentionné nulle part dans le fichier « EditeurTxTApp.cpp » un appel pour détruire le pointeur « leTexte ». Cela n'est pas nécessaire, car le parent de « wxTextCtrl », « TexteFrame » va se charger de détruire l'ensemble de ses enfants par relation quand il est détruit.

- Nous avons fait appel dans l'exemple à plusieurs méthodes définies dans « wxTextCtrl » comme « `SetDefaultStyle` » et « `AppendText` ». La première permet de définir le style par défaut alors que la seconde permet d'ajouter dans le contrôleur un texte.

- Pour le style par défaut, nous avons utilisé des instances de « `wxTextAttr` » et « `wxColour` ». Le premier représente les attributs d'un texte donné (couleur, taille, alignement, etc.), alors que le second représente une combinaison des trois couleurs (rouge, vert et bleu). Il permet de fixer la couleur du dessin.

4.2.3 Ajouter une barre de menus

- Nous allons ajouter des menus pour permettre aux utilisateurs de cette application de lire le contenu d'un fichier ou bien de préserver le texte modifié dans un fichier.

- Nous distinguons deux types de menus : « wxMenuBar » et « wxMenu ». La barre de menus contient les noms des menus de haut niveau. Dans cet exemple, nous allons ajouter deux menus de haut niveau « Fichier » et « Info ». Le menu « Fichier » va contenir 3 menus items : « Ouvrir », « Sauvegarder » et « Quitter ». Le menu « Info » contient l'item « Apropos ».

- Chaque menu nécessite un identificateur <u>unique</u> « ID ». Ces identificateurs sont utilisés pour associer les items avec leurs actions. La valeur de « ID » est un entier non nul.

```
// Fichier " EditeurTxTApp.h "

#ifndef EDITEURTXTAPP_H
#define EDITEURTXTAPP_H

enum {
    ID_QUITTER = 1,
    ID_OUVRIR = 100,
    ID_SAUVEGARDER = 200,
    ID_APROPOS = 300
};
```

```
class EditeurTxTApp : public wxApp {
public:
       virtual bool OnInit();
};

// Création de la classe associée à la fenêtre principale
// qui va contenir un titre, une bordure, etc.
class TexteFrame : public wxFrame {
public:
       // constructeur
       TexteFrame( const wxString titre, int xpos, int ypos,
                   int width, int height);
       // destructeur
       ~TexteFrame();
private:
        // " letexte " est un pointeur vers un contrôleur
        // de texte " wxTextCtrl "
        wxTextCtrl *leTexte;

        // une barre de menus : la barre de menus est
        // la pièce située en haut du cadre
        wxMenuBar *menuBar;


        // un menu : c'est un menu déroulant vers le bas.
        wxMenu *fichierMenu;
        wxMenu *infoMenu;
};
#endif
```


- Dans la classe « TextFrame » nous avons défini une barre de menus « wxMenuBar *menuBar » et deux menus de haut niveau « wxMenu *fichierMenu » et « wxMenu *infoMenu ».

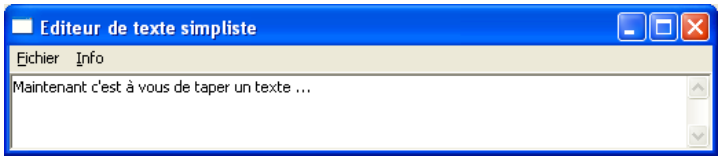
- Le seul changement notable dans le fichier « EditeurTxTApp.cpp » se situe au niveau du constructeur de classe « TexteFrame » comme suit :

```
// Fichier " EditeurTxTApp.cpp "

TexteFrame::TexteFrame
             (const wxString titre,
                    int xpos, int ypos,
                    int width, int height)
             : wxFrame
                ( (wxFrame *) NULL,
                   -1,
                   titre,
                   wxPoint(xpos, ypos),
                   wxSize(width, height)
                )
{


  // Initialisation du pointeur à la valeur "NULL castée"
  leTexte = (wxTextCtrl *) NULL;
  menuBar = (wxMenuBar *) NULL;
  fichierMenu = (wxMenu *) NULL;
```

```
        leTexte = new wxTextCtrl
                              ( this,
                                -1,
                                wxString("Maintenant c'est à vous
                                   de taper un texte ...\n\n"),
                                wxDefaultPosition,
                                wxDefaultSize,
                                wxTE_MULTILINE|wxTE_RICH2
                              );

        //Nous avons créé une instance de wxMenu pour « Fichier »

        fichierMenu = new wxMenu;

        // Nous utilisons les méthodes Append() et AppendSeparator()
        // pour ajouter des items et des séparateurs d'items au menu
        // « fichierMenu »

        fichierMenu->Append(ID_OUVRIR,  "&Ouvrir fichier");
        fichierMenu->Append(ID_SAUVEGARDER, "&Sauvegarder fichier");
        fichierMenu->AppendSeparator();
        fichierMenu->Append(ID_QUITTER,  "&Quitter");

        // Place à la création du menu « Apropos »

        infoMenu = new wxMenu();
        infoMenu->Append(ID_APROPOS, "&Apropos");
```



```
        // Pour finir nous allons créer une instance de wxMenuBar,
        // à laquelle nous allons associer les menus « Fichier »
        // et « Apropos ».
        // Nous déclarons les propriétés des menus dans un ordre
        // inverse. D'abord les items puis le menu et pour terminer
        // la barre de menus. C'est le principe des poupées russes.

        // Noter comment le « & » est utilisé dans les chaînes
        // "&Fichier" et "&Info". C'est un raccourci clavier
        // à un des éléments de la barre de menus ou les items
        // d'un menu donné.

        menuBar = new wxMenuBar;
        menuBar->Append(fichierMenu, "&Fichier");
        menuBar->Append(infoMenu, "&Info");

        // Nous utilisons par la suite la méthode « SetMenuBar() »
        // pour ajouter la barre de menus à notre cadre

        SetMenuBar(menuBar);

    }
```



4.2.4 Ajouter une barre d'état

- La barre d'état permet d'afficher des champs au bas de la « frame ». Ces champs sont sous la forme de chaînes de caractères.

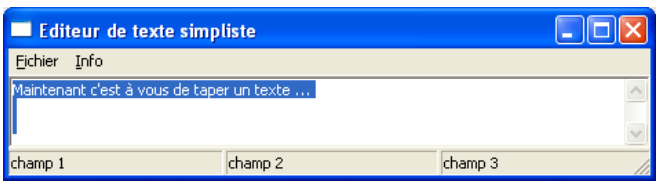
- Nous utilisons les méthodes : « CreateStatusBar » pour la création de la barre d'état d'un ou plusieurs champs ; « SetStatusText » pour initialiser un champ donné et « SetStatusWidths » pour calibrer la largeur du champ d'état.

- On n'apporte des modifications que dans le constructeur de la classe comme suit :

```
TexteFrame::TexteFrame
             (const wxString titre, int xpos, int ypos,
                    int width, int height)
             : wxFrame
                ( (wxFrame *) NULL,
                   -1, titre, wxPoint(xpos, ypos),
                   wxSize(width, height)
                )
{
  // [.....]
  // On ajoute ces lignes à la fin du constructeur

  // Nous allons ajouter à notre cadre, une barre
  // d'état contenant 3 champs

  CreateStatusBar(3);
```



```
  // Nous ajustons les valeurs des champs à l'aide
  // de la méthode SetStatusBarText("string", numero_du_champ)
  // de wxFrame. Dans cet exemple où la barre d'état a 3
  // champs, "index_champ" prend une des valeurs: 0, 1 ou 2.

  SetStatusText("champ 1", 0);
  SetStatusText("champ 2", 1);
  SetStatusText("champ 3", 2);
}
```



- L'utilisateur de l'application ne pourra pas gérer cette barre d'état puisque sa gestion est uniquement du ressort de la « frame ».

4.2.5 Gestion des événements des menus

- Vu que l'application a des menus, l'utilisateur aimerait bien les manipuler via le clavier ou la souris. Pour ce faire, le programme devra associer des traitements appropriés aux actions possibles de l'utilisateur.

- On distingue deux types d'événements :
  - événements de bas niveau comme appuyer ou relâcher un bouton de souris.
  - événements logiques comme cliquer sur une souris.

- Si vous appuyez par exemple sur la lettre « A », vous produisez les événements suivants :

  - 4 événements de bas niveau :
    - appuie sur la touche « shift »
    - appuie sur la touche « A »
    - relâchement de la touche « A »
    - relâchement de la touche « shift »

  - 1 événement logique :
    - frappe du caractère « A »

.

- Un événement reçu d'un menu est du type « wxCommandEvent ».

- L'idée consiste à déclarer dans la classe une méthode membre pour chaque évènement qui doit être traité. Il est d'usage de donner comme préfixe à ces méthodes le mot « On » (ou « A »).

- Afin de traiter tous les événements associés aux menus, nous allons les regrouper dans une table. Nous utilisons pour cela la directive « DECLARE_EVENT_TABLE ».

- L'implémentation de la table est codée dans le fichier « .cpp » comme suit :

```
BEGIN_EVENT_TABLE (NomClasse, NomClasseBase)
   EVT_TYPE (parametres, fonction)
   [....]
END_EVENT_TABLE()
```

- On peut avoir dans le programme plusieurs tables d'événements. Pour cette raison, nous devons préciser le nom de la classe dans la directive « BEGIN_EVENT_TABLE ».

- Afin d'attacher une méthode quelconque à un événement donné, nous utilisons la directive « EVT_TYPE ». Il y a plusieurs types d'événements: « EVT_MENU », « EVT_BUTTON », etc.

- Les modifications à apporter au fichier « `EditeurTxTApp.h` » sont comme suit :

```
// Fichier " EditeurTxTApp.h "


#ifndef EDITEURTXTAPP_H
#define EDITEURTXTAPP_H
```

```
class EditeurTxTApp : public wxApp {
public:
       virtual bool OnInit();
};

class TexteFrame : public wxFrame {
public:
       // constructeur
       TexteFrame( const wxString titre, int xpos, int ypos,
                   int width, int height);
       // destructeur
       ~TexteFrame();

       // Menu Fichier | Ouvrir
       void OnFichierOuvrir(wxCommandEvent &event);

       // Menu Fichier | Sauvegarder
       void OnFichierSauvegarder(wxCommandEvent &event);

       // Menu Fichier | Quitter
       void OnFichierQuitter(wxCommandEvent &event);

       // Menu Info | Apropos
       void OnInfoApropos(wxCommandEvent &event);

protected:
        DECLARE_EVENT_TABLE();
```



```
private:
        wxTextCtrl *leTexte;
        wxMenuBar *menuBar;
        wxMenu *fichierMenu;
        wxMenu *infoMenu;

        // Pour une question de design nous avons inclus
        // directement dans la classe les identificateurs
        // de menu. Le premier va prendre la valeur 100,
        // le 2nd va avoir la valeur 101, le 3e 102 et le 4e
        // 103, à cause des propriétés de « enum ».

        enum {
             ID_QUITTER = 100,
             ID_OUVRIR,
             ID_SAUVEGARDER,
             ID_APROPOS
        };

};
#endif
```

- Nous ajoutons les lignes de code ci-dessous à la fin du fichier « `EditeurTxTApp.cpp` » comme suit :

```
// Fichier " EditeurTxTApp.cpp "
// [……]

BEGIN_EVENT_TABLE(TexteFrame, wxFrame)
    EVT_MENU(ID_OUVRIR, TexteFrame::OnFichierOuvrir)
    EVT_MENU(ID_SAUVEGARDER, TexteFrame::OnFichierSauvegarder)
    EVT_MENU(ID_QUITTER, TexteFrame::OnFichierQuitter)
    EVT_MENU(ID_APROPOS, TexteFrame::OnInfoApropos)
END_EVENT_TABLE()

void TexteFrame::OnFichierOuvrir(wxCommandEvent &event){
    wxLogMessage("Fichier|Ouvrir");
}

void TexteFrame::OnFichierSauvegarder(wxCommandEvent &event){
    wxLogMessage("Fichier|Sauvegarder");
}

void TexteFrame::OnFichierQuitter(wxCommandEvent &event){
    Close(FALSE);
}

void TexteFrame::OnInfoApropos(wxCommandEvent &event){
    wxLogMessage("Info|Apropos");
}

// Fin fichier " EditeurTxTApp.cpp "
```


4.3. Utilisation des « Dialog »

- Une boîte de dialogue est une fenêtre avec une barre de titre.

- La fenêtre peut être déplacée et peut contenir des contrôleurs et d'autres fenêtres.

- Nous distinguons deux types de dialogue : modal (en anglais « modal ») et non modal (en anglais « modeless »).

- Une boîte de dialogue modal est une boîte qui, en attente d'une réponse, bloque l'accès à votre interface graphique tant qu'elle n'a pas reçu de réponse. Un exemple est la boîte « sauvegarder sous » qui, lorsque vous faites appel, vous ne pouvez plus accéder à la fenêtre parente tant que vous n'avez pas effectué la sauvegarde ou bien annulé l'opération de sauvegarde.

- Une boîte de dialogue non modal est tout le contraire d'une boîte de dialogue modal. Même après avoir lancé cette boîte de dialogue, l'accès à la fenêtre parente reste possible. Un exemple est le gestionnaire de favoris dans un navigateur. Le navigateur va ouvrir une fenêtre à part pour gérer les favoris et il vous permet en parallèle d'accéder à la fenêtre parente ici le navigateur.

- Dans l'application nous avons deux menus items pour l'ouverture et la sauvegarde de fichiers. Nous pouvons utiliser dans ce cas la fenêtre de dialogue « wxFileDialog ».

- Quand une boîte de dialogue est utilisée, elle doit être d'un look familier à l'usager du système. Par exemple : pour ouvrir un fichier sous Windows nous allons faire appel « wxFileDialog » qui fera appel à la fenêtre de dialogue de Windows, car l'usager est plus familier avec cette dernière. Il en sera de même sur un autre système.

- wxWindows fournit aux programmeurs un certain nombre de boîtes de dialogue parmi lesquelles :

Classe (boîte de dialogue)	Description
wxColourDialog	Affiche une boîte de dialogue pour choisir une couleur
wxFontDialog	Affiche une boîte de dialogue pour choisir une fonte
wxPrintDialog	Affiche une boîte de dialogue pour configurer une imprimante ou imprimer un document
wxFileDialog	Affiche une boîte de dialogue pour la sauvegarde ou l'ouverture d'un fichier
wxDirDialog	Affiche une boîte de dialogue pour choisir un répertoire
wxTextEntryDialog	Affiche une boîte de dialogue pour demander à l'usager d'insérer une seule ligne de texte
wxMessageDialog	Affiche une boîte de dialogue avec un message sur une ligne ou plusieurs lignes avec les choix suivants : Ok, Yes (Oui), No (Non) et Cancel (Annuler)
wxSingleChoiceDialog	Affiche une boîte de dialogue avec une liste de chaînes et l'usager doit en sélectionner une

4.3.1 « wxFileDialog »

- Cette boîte de dialogue présente à l'usager la possibilité de choisir un nom quand il veut ouvrir ou sauvegarder un fichier.

- Le constructeur de « wxFileDialog » est comme suit :

```
wxFileDialog(wxWindow* parent,
             const wxString& message = "Choisir un fichier",
             const wxString& RepParDefaut = "",
             const wxString& FichierParDefaut = "",
             const wxString& joker = "*.*",
             long style = 0,
             const wxPoint& pos = wxDefaultPosition);
```

- « parent » est la fenêtre qui détient ce dialogue.

- « message » est le titre du dialogue.

- « RepParDefaut » est le nom du répertoire où la boîte de dialogue doit se positionner par défaut.

- « FichierParDefaut » est le nom du fichier à utiliser par défaut.

- « joker » (« wildcard » en anglais » est quelque chose comme « *.* » ou bien « *.txt ». Ce paramètre est utilisé pour filtrer les réponses affichées par la boîte de dialogue. Il est possible de lister plusieurs

extensions à la fois en utilisant la syntaxe « description|extension » comme suit : « Tous les fichiers(*.*)|*.*|Fichiers Textes(*.txt)|*.txt|Fichiers Bitmap(*.bmp)|*.bmp ».

- « style » est le type de la boîte de dialogue qui peut-être :

Style	Description
wxFD_OPEN	Utiliser pour l'ouverture
wxFD_SAVE	Utiliser pour la sauvegarde
wxHide_READONLY	Utiliser pour cacher les fichiers en mode de lecture uniquement
wxOVERWRITE_PROMPT	Utiliser pour demander une confirmation quand un fichier sera écrasé
wxMULTIPLE	Utiliser pour l'ouverture uniquement. Elle permet la sélection de plusieurs fichiers à la fois

- « pos », option non utilisée.

- Nous allons modifier le cœur des deux méthodes « OnFichierOuvrir » et « OnFichierSauvegarder » du fichier « `EditeurTxTApp.cpp` » comme suit :

```
// Fichier " EditeurTxTApp.cpp "
// [……]

void TexteFrame::OnFichierOuvrir(wxCommandEvent &event){
     wxFileDialog *dlg = new wxFileDialog(this,
                         "Ouvrir un fichier texte", "", "",
                         "Tous les fichiers(*.*)|*.*|Fichiers
                         textes(*.txt)|*.txt", wxFD_OPEN,
                         wxDefaultPosition);
     if ( dlg->ShowModal() == wxID_OK )
        leTexte->LoadFile(dlg->GetFilename());
     dlg->Destroy();
}
void TexteFrame::OnFichierSauvegarder(wxCommandEvent &event){
     wxFileDialog *dlg = new wxFileDialog(this,
                         "Sauvegarder un fichier texte", "", "",
                         "Tous les fichiers(*.*)|*.*|Fichiers
                         textes(*.txt)|*.txt", wxFD_SAVE,
                         wxDefaultPosition);

     if ( dlg->ShowModal() == wxID_OK )
        leTexte->SaveFile(dlg->GetPath());

     dlg->Destroy();
}
// Fin fichier " EditeurTxTApp.cpp "
```

- Nous avons précisé dans la première méthode que la boîte de dialogue doit être du type « wxFD_Open » et dans la seconde méthode du type « wxFD_Save ».

- Dans les deux méthodes, nous avons fait appel à la méthode « ShowModal ». Ceci signifie qu'aucune autre fenêtre ne peut obtenir le focus. Nous avons donc créé des boîtes de dialogue modal. Cette méthode retourne « wxID_OK » si l'usager a appuyé sur le bouton « OK » et « wxID_CANCEL » dans le cas contraire.

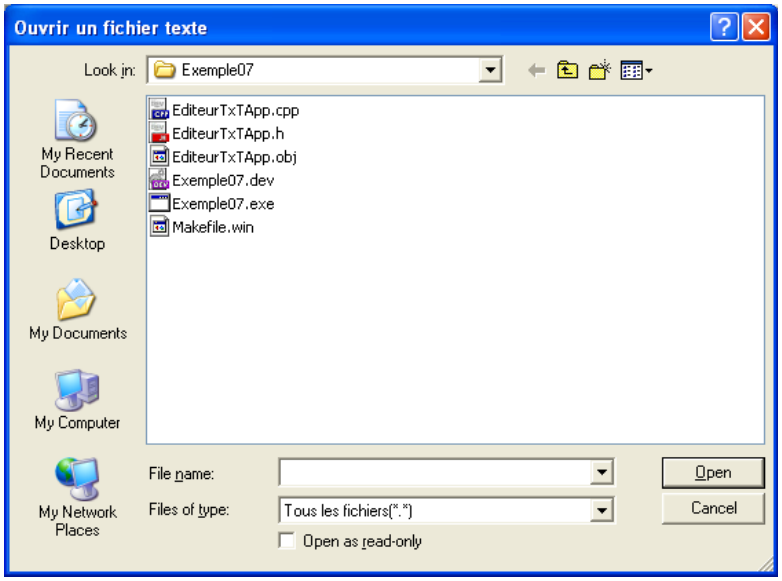
- La méthode « GetFilename » récupère le nom du fichier sélectionné lors de l'opération d'ouverture de fichier, alors que la méthode « GetPath » va retourner le chemin complet (nom fichier et nom du répertoire) du fichier sélectionné.

- Nous utilisons les méthodes « LoadFile » et « SaveFile » définies dans « wxTextCtrl » pour charger ou sauvegarder le fichier.

- Pour chaque boîte de dialogue, nous avons fait appel à la méthode « Destroy » au lieu de détruire les pointeurs directement. Ceci pour des raisons de consistance. « wxWidgets » retarde ainsi la destruction des boîtes de dialogue tant qu'il n'a pas terminé le traitement de tous les événements les affectant. « wxWidgets » évite ainsi d'envoyer des requêtes à des boîtes de dialogue déjà détruites.

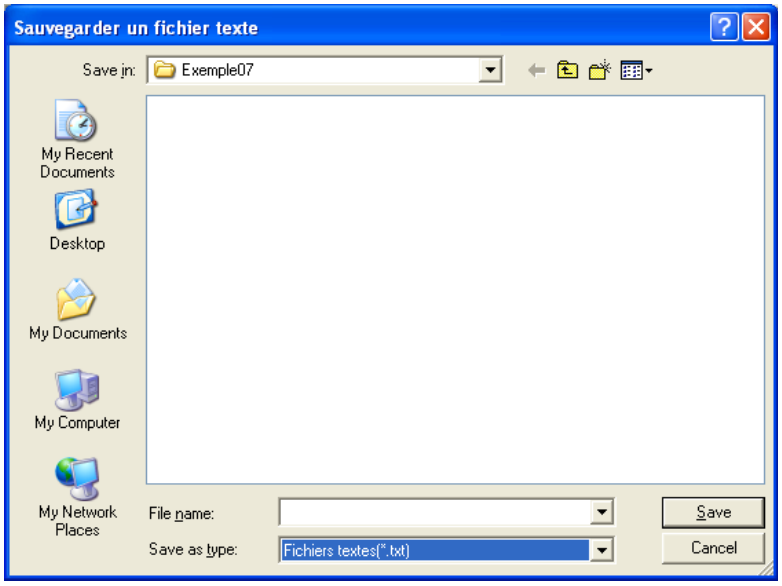
- Dans les deux opérations décrites ci-dessous, nous constatons que dans les deux cas « wxWidgets » a fait appel aux interfaces de Windows responsables de l'ouverture ou de la sauvegarde de fichiers. Ainsi l'usager d'un tel système n'est pas dépaysé quand il rencontre ce genre de boîtes de dialogue.

- Nous obtenons les résultats suivants dans le cas d'une ouverture d'un fichier :



- On constate que le filtre est à « *.* », ce qui a permis d'afficher tous les fichiers qui se trouvent dans le répertoire courant de l'application.

- Nous obtenons les résultats suivants dans le cas d'une sauvegarde d'un fichier :



- Dans ce cas nous avons opté pour le filtre est « *.txt ». Comme le répertoire ne contient aucun fichier texte, la fenêtre ne propose aucun fichier.

4.3.2 « wxMessageDialog »

- Affiche une boîte de dialogue avec un message sur une ligne ou plusieurs lignes avec les choix suivants : Ok, Yes (Oui), No (Non) et Cancel (Annuler).

- Le constructeur de « wxMessageDialog » est comme suit :

```
wxMessageDialog(wxWindow* parent, const wxString& message,
                const wxString& titre = "Message box",
                long style = wxOK | wxCANCEL | wxCENTRE,
                const wxPoint& pos = wxDefaultPosition);
```

- « parent » est la fenêtre parente de ce dialogue.

- « message » est le message affiché. Il peut être sur plusieurs lignes.

- « titre » est le titre associé à la fenêtre dialogue.

- « style » est le style du dialogue. Les styles possibles sont :

Style	Description
wxOK	Permet d'afficher le bouton « OK »
wxCANCEL	Permet d'afficher le bouton « CANCEL »
wxYES_NO	Permet d'afficher les boutons « YES » et « NO »
wxYES_DEFAULT	Permet d'afficher les boutons « YES » et « NO ». Le bouton « YES » est marqué par défaut. C'est le comportement par défaut.
wxNO_DEFAULT	Permet d'afficher les boutons « YES » et « NO ». Le bouton « NO » est marqué par défaut.
wxCentre	Permet de centrer le message.
wxICON_EXCLAMATION	Permet d'afficher un point d'exclamation
wxICON_HAND	Permet d'afficher une erreur
wxICON_ERROR	Permet d'afficher une erreur (idem que wxICON_HAND)
wxICON_QUESTION	Permet d'afficher un point d'interrogation
wxICON_INFORMATION	Permet d'afficher une information

- « pos » est la position du dialogue.

- Nous allons modifier le cœur de la méthode « OnInfoApropos » du fichier « `EditeurTxTApp.cpp` » comme suit :


```
// Fichier " EditeurTxTApp.cpp "
// [……]

void TexteFrame::OnInfoApropos(wxCommandEvent &event){
     wxMessageDialog *dlg = new wxMessageDialog(this,
            "Éditeur de texte simpliste créé avec wxWidgets.",
            "Message de dialogue",
            wxOK | wxCENTRE | wxICON_INFORMATION,
            wxDefaultPosition);
     dlg->ShowModal();
     dlg->Destroy();
}
// Fin fichier " EditeurTxTApp.cpp "
```

4.3.3 « wxTextEntryDialog »

- C'est une boîte de dialogue qui réclame de l'usager d'insérer une seule ligne de texte

- Le constructeur de « wxEntryDialog » est comme suit :

```
wxTextEntryDialog(wxWindow *parent, const wxString& message,
                const wxString& titre = "svp insérer un texte",
                const wxString& ValeurParDefaut,
                long style = wxOK | wxCANCEL | wxCENTRE,
                const wxPoint& pos = wxDefaultPosition);
```

- « parent » est la fenêtre parente de ce dialogue.
- « message » est le message affiché. Il peut être sur plusieurs lignes.
- « titre » est le titre associé à la fenêtre dialogue.
- « ValeurParDefaut » est le texte par défaut.
- « style » est le style du dialogue. Tous les styles de « wxTextCtrl » peuvent être utilisés à ce niveau.
- « pos » est la position par défaut.
- Nous allons modifier le cœur de la méthode « OnInfoApropos » du fichier « `EditeurTxTApp.cpp` » comme suit :

```
wxTextEntryDialog *dlg = new wxTextEntryDialog(this,
                "Entrer votre mot de passe",
                "Entrer votre mot de passe", "",
                wxOK | wxCANCEL | wxCENTRE | wxTE_PASSWORD);

if ( dlg->ShowModal() == wxID_OK ){
    // Vérifier le mot de passe
}else{
    // Arrêter l'application
}
```


4.3.4 « Boîtes de dialogues personnalisées »

- « wxDialog » est la classe de base de tous les dialogues.
- La classe contenant une boîte de dialogues personnalisés doit dériver de cette classe de base.
- Le constructeur de « wxDialog » est comme suit :

```
wxDialog(wxWindow* parent, wxWindowID id, const wxString& title,
              const wxPoint& position = wxDefaultPosition,
              const wxSize& taille = wxDefaultSize,
              long style = wxDEFAULT_DIALOG_STYLE,
              const wxString& nom = "BoîteDeDialogue");
```

- « parent » est la fenêtre parente de ce dialogue.
- « Id » est l'identificateur unique de la fenêtre. Mettre « - 1 » pour la valeur par défaut.
- « titre » est le titre associé à la fenêtre dialogue.
- « position » est une position relative à la fenêtre parente.
- « taille » les dimensions de la fenêtre de dialogue.

- « style » est le style du dialogue. Tous les styles de « wxTextCtrl » peuvent être utilisés à ce niveau.
- « nom » est le nom associé à la fenêtre.
- Nous allons modifier le cœur de la méthode « OnInfoApropos » du fichier « `EditeurTxTApp.cpp` » comme suit :

```
void TexteFrame::OnInfoApropos(wxCommandEvent &event){
  TestDialogue
   aproposDialogue ( this,
                     -1,
                     "Votre boîte de dialogue personnalisée",
                     wxPoint(100,100),
                     wxSize(200,200)
                   );
  if (aproposDialogue.ShowModal() != wxID_OK)
   leTexte->AppendText("Vous avez cliqué sur
                                  le bouton <annuler>.\n");
  else
   leTexte->AppendText(aproposDialogue.GetText());
}
```


- Nous obtenons le résultat suivant :

- Nous allons décrire dans les prochains paragraphes les éléments supplémentaires utilisés pour concevoir cette boîte de dialogue. Parmi ces éléments on peut citer : les boutons (« OK » et « Annuler »), la disposition de ces boutons dans la boîte de dialogue, etc.

4.3.5 Fichier de ressources « rc »

- Un programme peut faire appel à des données externes comme des images, des icônes, des chaînes de caractères, des menus, etc.
- On regroupe ces données dans un fichier des ressources ayant l'extension « .rc ».
- La notion de fichier de ressources est propre à Windows.

- Cette manière de procéder permet de séparer les données externes du code de l'application.
- Quand vous développez une boîte de dialogue personnalisée, « wxWidgets » a généralement besoin sous Windows d'un fichier de ressources.
- Ce fichier doit contenir au moins la ligne suivante :

```
// Fichier " EditeurTxTApp.rc "

#include "wx/msw/wx.rc"
```

- Ce fichier sera compilé et ajouté au moment de l'édition de liens au programme exécutable.
- Le fichier de ressource peut contenir d'autres appels, par exemple :

```
// Fichier " EditeurTxTApp.rc "

#include "wx/msw/wx.rc"

exempleicone ICON "computer.ico"
```

- La ligne est composée de 3 éléments : un identificateur (« exempleicon »), le type de la ressource (ici « ICON ») et le contenu de la ressource (ici le fichier « computer.ico »).

- Le nom du fichier à inclure doit être accessible au compilateur sinon il faudra inclure le chemin complet.
- Pour plus de détails, voir l'API Windows.
- L'appel de la ressource (ici une icône) dans le programme se fait comme suit :

```
// Fichier " EditeurTxTApp.rc "

TexteFrame::TexteFrame ( … … … … …) : wxFrame( …… ){

  // … … … …
  SetIcon(wxICON(exempleicone));
  // … … … …
}
```

- « wxWidgets » utilise maintenant un nouveau standard pour la gestion des ressources.
- Ce standard utilise le protocole « XML ».
- Les fichiers de ressources ont dans ce cas l'extension « xrc ».

4.4. Positionnement des « wxWidgets »

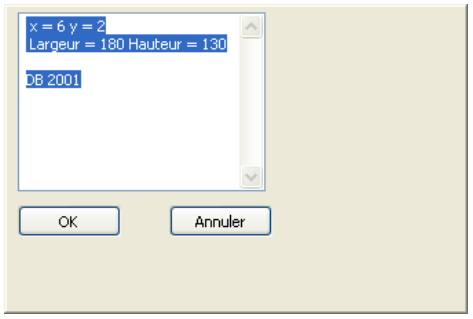
- Une interface graphique peut-être constituée de plusieurs « wxWidgets ».

- Il faudra donc trouver un moyen pour disposer correctement les « wxWidgets » dans l'interface afin de tenir compte des dimensionnements possibles.

- Dans l'exemple précédent, si on avait autorisé l'usager à redimensionner l'interface, nous aurions modifié l'appel au constructeur de la boîte de dialogue en ajoutant l'option « wxRESIZE_BORDER » comme suit :

```
void TexteFrame::OnInfoApropos(wxCommandEvent &event){
  TestDialogue
   aproposDialogue ( this,
                  -1,
                  "Votre boîte de dialogue personnalisée",
                  wxPoint(100,100),
                  wxSize(200,200),
                  wxRESIZE_BORDER
                );
  if (aproposDialogue.ShowModal() != wxID_OK)
   leTexte->AppendText("Vous avez cliqué sur
                                le bouton <annuler>.\n");
  else
   leTexte->AppendText(aproposDialogue.GetText());
}
```


- En redimensionnant la fenêtre, on se retrouve avec de l'espace vide tout autour des boutons.



Après avoir redimensionné la fenêtre de dialogue.

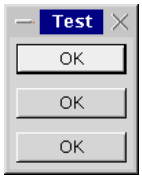
- On pouvait recalculer les nouvelles positions des boutons et introduire ces changements lorsque l'usager a redimensionné la fenêtre.

- Même si cette approche n'est pas complexe à coder, elle génère par contre un code volumineux. La raison pour cela : il suffit de redimensionner l'interface graphique, que vous êtes obligés de refaire les calculs pour faire apparaître correctement cette interface graphique.

- On peut se servir d'une autre approche qui utilise des gestionnaires d'espace. Ces gestionnaires sont appelés des « sizers ».

- La mise en page des composants est ainsi déléguée à des « sizers ».

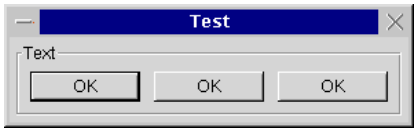
- Les « sizers » dérivent de la classe « wxSizer » et permettent de ranger les « widgets » qu'ils contiennent.
- Le principe des « sizers » est comme suit :
    - On construit d'abord une boîte invisible en précisant le type d'alignement souhaité dans cette boîte (alignement vertical, alignement horizontal, etc.).
    - On place par la suite les composants graphiques en indiquant comment ils doivent s'agencer entre eux.
- Les éléments communs :
    - Taille minimale, bordure, l'alignement, l'étirement et le facteur de compensation.
- Les différents « sizers » :
    - « wxBoxSizer » : les « widgets » sont rangées les unes à la suite des autres horizontalement ou verticalement.



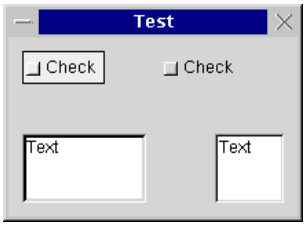
verticalement

horizontalement

- « wxStaticBoxSizer » : il a le même comportement que « wxBoxSizer » sauf qu'il entoure les « widgets » avec une bordure.

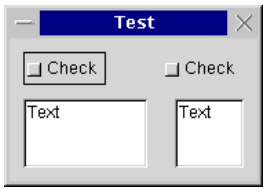


- « wxGridSizer » : il range les « widgets » sur deux dimensions. Tous les enfants ont la même taille, celle du plus grand. Soit le nombre de lignes ou bien le nombre de colonnes sont fixés. L'ajout d'un enfant se fait dans l'axe non fixé.



- « wxFlexGridSizer » : il range les « widgets » sur deux dimensions. La largeur et la hauteur de chaque cellule sont calculées de manière individuelle. Ce calcul va tenir compte de la

taille de la « widget » contenue dans la cellule. Chaque cellule peut-être autorisée à être étirée.




```
// Construction d'un "sizer" vertical

wxBoxSizer *dialogSizer = new wxBoxSizer(wxVERTICAL);

// Construction d'un "grid sizer" avec 2 lignes et 2 colonnes
// on laisse 10 pixels d'espace entre les lignes et 10 pixels
// d'espace entre les colonnes

wxFlexGridSizer *textSizer = new wxFlexGridSizer(2, 2, 10, 10);
```

- La méthode « Add » définie dans « wxSizer » permet d'ajouter des éléments dans un « sizer ». Ces éléments peuvent être des contrôleurs, d'autres « sizers » ou des espaces.

- Ajouter des contrôles :


```
void Add(wxWindow* window, int option = 0, int flag = 0,
         int bordure = 0, wxObject* userData = NULL)
```

- « window » est le contrôleur à ajouter à la fenêtre (un bouton, un texte, etc.).

- « option » est utilisée avec « wxBoxSizer ». Quand elle prend la valeur « 0 » cela signifie la taille du contrôleur ne va pas être modifiée dans l'orientation principale du « sizer » (c.-à-d. de manière horizontale ou verticale). « 1 » dans le cas contraire.

- « flag » une série de volets qui peuvent être combinés moyennant l'opérateur « OR » « | ». On distingue deux types de volets :

  - volets de bordure

wxTOP	La bordure est située en haut
wxBOTTOM	La bordure est située en bas
wxLEFT	La bordure est située à gauche
wxRIGHT	La bordure est située à droite
wxALL	La bordure est située sur tous les côtés

- volets de comportement

wxGROW ou wxEXPAND	L'item peut-être redimensionné
wxSHAPED	L'item peut-être redimensionné de manière proportionnelle
wxALIGN_CENTER ou wxALIGN_CENTRE	L'item est aligné au centre
wxALIGN_LEFT	L'item est aligné à gauche
wxALIGN_TOP	L'item est aligné vers le haut
wxALIGN_RIGHT	L'item est aligné à droite
wxALIGN_CENTER_HORIZONTAL	L'item est centré dans l'orientation principale horizontale
wxALIGN_CENTER_VERTICAL	L'item est centré dans l'orientation principale verticale

- « bordure » elle représente la largeur de la bordure quand le volet bordure est utilisé.
- « userData » elle permet l'attachement d'un autre objet.

- Ajouter des « sizers » :

```
void Add(wxSizer* sizer, int option = 0, int flag = 0,
         int bordure = 0, wxObject* userData = NULL)
```

- Dans la méthode « Add », le premier argument est un « wxSizer » dans ce cas.

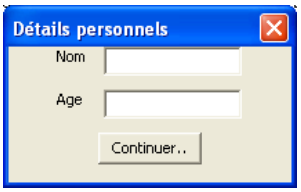
- Les « sizers » peuvent être imbriqués.

- Ajouter des « espaces » :
  - Les espaces sont introduits entre des contrôleurs pour les séparer d'une distance donnée.
  - On peut par exemple ajouter des espaces entre deux boutons.

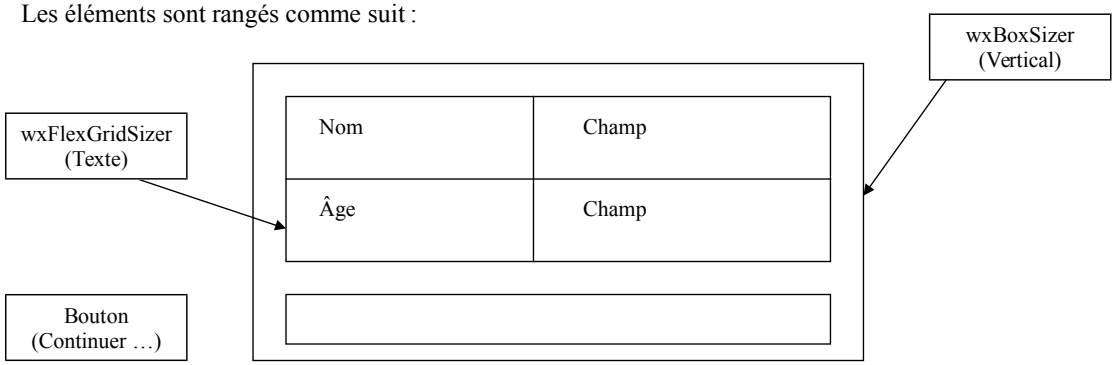
```
void Add(int largeur, int hauteur, int option = 0,
         int flag = 0, int bordure = 0,
         wxObject* userData = NULL)
```

- Dans la méthode « Add » nous avons introduit les deux arguments « int largeur » et « int hauteur ».

- Nous allons manipuler dans notre exemple deux types de « sizers » : « wxBoxSizer » et « wxFlexGridSizer » afin d'obtenir l'interface graphique suivante :



- Les éléments sont rangés comme suit :



  - Nous créons d'abord un « wxBoxSizer » avec une orientation verticale. Tous les enfants seront insérés de manière verticale. Ils ne peuvent être redimensionnés que de manière verticale.

```
wxBoxSizer *dialogSizer = new wxBoxSizer(wxVERTICAL);
```

  - Nous créons par la suite un « wxFlexGridSizer » de 2 lignes et 2 colonnes afin d'ajouter 2 étiquettes (« Nom » et « Âge ») et 2 contrôleurs de texte (des champs). Nous obtenons ainsi 4 cellules.


```
nom = new wxTextCtrl(this, -1);
age = new wxTextCtrl(this, -1);

wxFlexGridSizer *texteSizer =
                    new wxFlexGridSizer(2, 2, 10, 10);

textSizer->Add(new wxStaticText(this, -1, "Nom "),
                    1, // étendre verticalement
                    wxEXPAND); // étendre horizontalement

texteSizer->Add(nom, 1, wxEXPAND);

texteSizer->Add(new wxStaticText(this, -1, "Age "),
                    1, wxEXPAND);

texteSizer->Add(age, 1, wxEXPAND);
```

  - Par la suite c'est le tour de créer le bouton comme suit :

```
wxButton *button =
        new wxButton(this, ID_BUTTON_CLICK, "Continuer..");
```

- Nous allons ajouter au « wxBoxSizer » les « wxFlexGridSizer » et le bouton comme suit :

```
dialogSizer->Add(texteSizer, 0, wxALIGN_CENTER);

dialogSizer->Add(button,
          0, // impossible de l'étendre verticalement
          wxALIGN_CENTER | wxALL, 10); // centre
                                 // horizontalement et fixe
                                 // la bordure à 10 pixels
```

- Finalement :

```
// On a choisi "dialogSizer" pour positionner les éléments
SetSizer(dialogSizer);

// permet de donner la taille exacte nécessaire pour
// afficher les composants
dialogSizer->Fit(this);
```

Pour plus d'informations sur le positionnement, voir cette page web :

What Do These Sizer Things Do?