Cours wxWidgets : 2ème partie

Cours wxWidgets : 2ème partie....1
1) Introduction....1
2) Calculer la somme MD5 d'un fichier....2
3) Le composant personnalisé autonome....2
4) La classe d'application et la fenêtre principale....8
5) Accepter le glisser/déposer de fichiers....11
6) Demander à la fenêtre principale de comparer les sommes MD5....13
7) Conclusion....15

1) Introduction

Nous voici donc de nouveau ensembles pour la deuxième partie de ce cours sur wxWidgets.

Nous allons, tout au long de cette partie, créer un petit utilitaire permettant de calculer la somme MD5 d'un fichier.

Ensuite, nous ferons en sorte de pouvoir le faire pour deux fichiers, et ainsi, pouvoir comparer les deux sommes calculées et indiquer à l'utilisateur si elles sont identiques ou non.

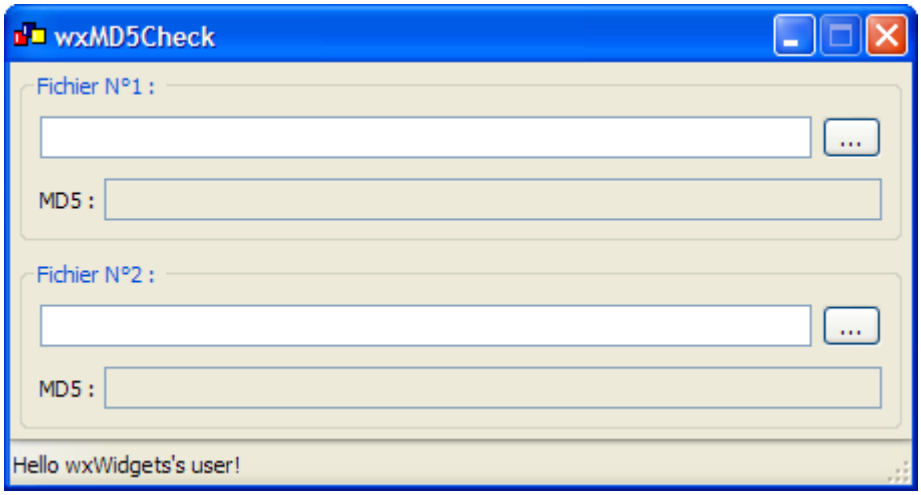
Pour sélectionner le ou les fichiers dont il voudra calculer la somme MD5, l'utilisateur aura plusieurs possibilités :

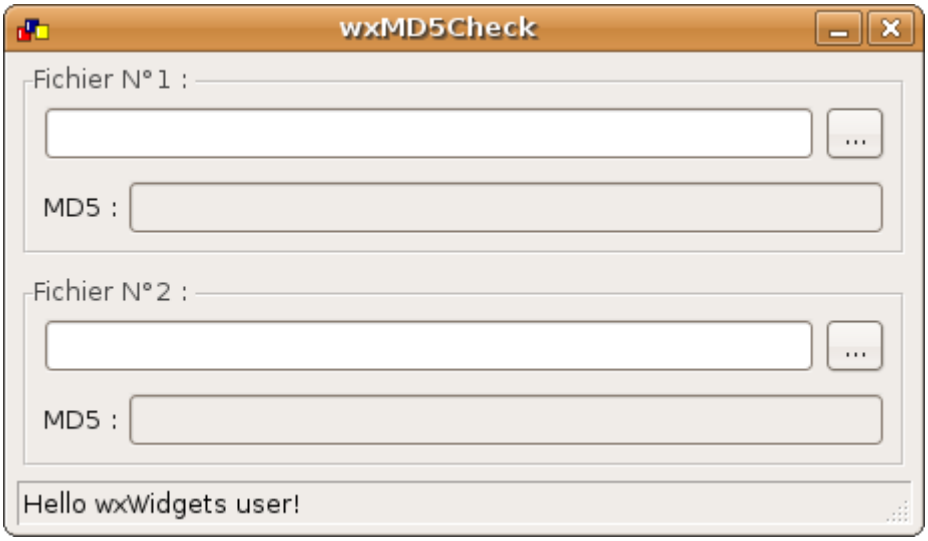
- saisir le chemin complet du fichier dans une zone de texte
- cliquer sur un bouton « parcourir » et ainsi sélectionner le fichier graphiquement
- faire glisser le fichier dans la zone de texte pour qu'on puisse en récupérer le chemin

Pour pouvoir ajouter facilement la possibilité de traiter deux fichiers, nous allons créerun composant personnalisé autonome. De ce fait, nous n'aurons qu'à décider si nous l'ajoutons une ou plusieurs fois dans la fenêtre principale, et le tour est joué.

Il restera ensuite à créer un événement personnalisé pour que la fenêtre soit informée que l'une des deux valeurs a changé, et qu'il faut la comparer avec l'autre pour indiquer à l'utilisateur si elles sont identiques ou différentes.

Voici une capture, sous Windows et sous Ubuntu, de l'application telle qu'elle sera lorsqu'elle sera terminée.





Dernière petite précision : je ne m'attarderais pas sur la création du projet et sur les réglages éventuels de l'IDE : assurez-vous d'être en mesure de le faire par vous-même (vous pouvez bien entendu revoir la première partie du cours, ou vous faire aider sur wxDev.fr).

2) Calculer la somme MD5 d'un fichier

Pour ce chapitre du cours, nous n'allons pas réinventer la roue mais tout simplement utiliser un code trouvé sur internet. Je ne vais pas vous en fournir le listing ici, mais vous trouverez les fichiers concernés dans le projet complet que vous trouverez sur wxDev.fr.

Il se compose d'un fichier header et d'un fichier source en C.

Voici la méthode à suivre pour calculer la somme MD5 d'un fichier :

- initialiser une zone tampon de type **`md5_byte_t`** et de taille **`512`**
- créer une variable de type **`md5_state`** et l'initialiser au moyen d'une fonction créée pour cela.
- lire 512 octets du fichier, les placer dans le tampon ci-dessus, calculer la somme, et ainsi de suite jusqu'à la fin du fichier
- demander le résultat final

3) Le composant personnalisé autonome

Comme je vous l'ai dit précédemment, nous allons créer un composant personnalisé autonome qui sera capable de recevoir un fichier par glisser-déposer, qui proposera un bouton permettant d'aller sélectionner le fichier à traiter à l'aide d'une boite de dialogue, qui calculera la somme MD5 du fichier sélectionné et qui l'affichera.

Voici à quoi il ressemble sous Windows lorsqu'il est isolé du reste de l'application

Ce composant sera dérivé de la classe **`wxPanel`** qui est un simple contrôle de base.

Nous y placerons un premier sizer classique (un **`wxBoxSizer`**) qui nous permettra de mettre une « marge » entre les bords du panel et le cadre. Il pourra être vertical ou horizontal, peu importe, car il ne contiendra qu'un seul contrôle.

Nous placerons ensuite un **`wxStaticBoxSizer`** vertical qui nous permettra de créer le cadre (dont le titre sera d'ailleurs passé en paramètre lors de la construction du composant) ainsi que de gérer les deux lignes de contrôles.

La première ligne sera gérée par un sizer classique (un **wxBoxSizer**) horizontal dans lequel nous placerons une zone de texte qui contiendra le chemin du fichier à traiter ainsi qu'un bouton permettant l'affichage de la boite de dialogue pour la sélection du fichier.

La deuxième ligne sera également gérée par un sizer horizontal qui contiendra un texte statique, et une zone de texte en lecture seule dans laquelle nous afficherons le résultat.

Voici tout d'abord le code de base de notre composant personnalisé : nous remplirons les différentes méthodes par la suite.

Fichier « File2CheckPanel.h »

```
#ifndef FILE2CHECKPANEL_H_INCLUDED
#define FILE2CHECKPANEL_H_INCLUDED

#include <wx/wx.h>

class File2CheckPanel : public wxPanel
{
    public:
        File2CheckPanel(wxWindow* parent, const wxString& title);
        virtual ~File2CheckPanel();
    private:
        wxTextCtrl *m_txtFileName, *m_txtResult;
        wxButton *m_btnBrowse;
        void CreateAndConnectControls(const wxString& title);
};

#endif // FILE2CHECKPANEL_H_INCLUDED
```

Vous constaterez la présence d'une méthode privée « **CreateAndConnectControls** » permettant d'isoler le code correspondant à la création des contrôles contenus pas le composant ainsi que les variables servant à recevoir les pointeurs des contrôles.

Voici maintenant le code source de base :

Fichier « File2CheckPanel.cpp »

```
#include "File2CheckPanel.h"

File2CheckPanel::File2CheckPanel(wxWindow* parent, const wxString& title)
    : wxPanel(parent, -1)
{
    // On défini une taille minimale pour que ça ressemble à quelque chose
    SetMinSize(wxSize(450,-1));
    // On appelle la méthode servant à la création et à la connexion des contrôles
    CreateAndConnectControls(title);
}

File2CheckPanel::~File2CheckPanel()
{
    //dtor
}

void File2CheckPanel::CreateAndConnectControls(const wxString& title)
{
    // création et connexion des contrôles
}
```

Nous allons maintenant voir le code servant à la création des contrôles :

```
  void File2CheckPanel::CreateAndConnectControls(const wxString& title)
  {
      // Création du sizer principal
      wxBoxSizer* mainsizer=new wxBoxSizer(wxVERTICAL);
          // Le wxStaticBoxsizer servant à gérer les deux lignes de contrôles
          wxStaticBoxSizer* box=new wxStaticBoxSizer(wxVERTICAL, this, title);
              // Le wxBoxSizer pour la première ligne
              wxBoxSizer* line1=new wxBoxSizer(wxHORIZONTAL);
                  // Le wxTextCtrl servant à recevoir le chemin du fichier à traiter
                  m_txtFileName=new wxTextCtrl(this, -1, _T(""));
                  // On l'ajoute au sizer avec une proportion de 1 afin qu'il
                  // soit étendu lors du redimensionnement
                  line1->Add(m_txtFileName, 1, wxALL|wxALIGN_CENTER_VERTICAL, 0);
                  // On ajoute un espace de 5 pixels avant le contrôle suivant
                  line1->AddSpacer(5);
                  // Le bouton pour la boite de dialogue de sélection du fichier
                  m_btnBrowse=new wxButton(this, -1, _T("..."), wxDefaultPosition,
wxSize(30,15));
                  // On l'ajoute au sizer avec une proportion de 0 car il
                  // n'a pas besoin d'être redimensionné
                  line1->Add(m_btnBrowse, 0, wxALL|wxEXPAND, 0);
              // On ajoute la première ligne au wxStaticBoxsizer
              box->Add(line1, 0, wxALL|wxEXPAND, 5);
              // Le wxBoxSizer pour la deuxième ligne
              wxBoxSizer* line2=new wxBoxSizer(wxHORIZONTAL);
                  // Le texte statique
                  wxStaticText* label=new wxStaticText(this, -1, _T("MD5 : "));
                  // On l'ajoute au sizer avec une proportion de 0
                  line2->Add(label, 0, wxALL|wxALIGN_CENTER_VERTICAL, 0);
                  // La zone de texte en lecture seule pour le résultat
                  m_txtResult=new wxTextCtrl(this, -1, _T(""), wxDefaultPosition,
wxDefaultSize, wxTE_READONLY);
                  // On modifie la couleur de fond pour indiquer à l'utilisateur
                  // que cette zone n'est pas éditable
                  m_txtResult-
>SetBackgroundColour( wxSystemSettings::GetColour( wxSYS_COLOUR_BTNFACE));
                  // On l'ajoute au sizer avec une proportion de 1
                  line2->Add(m_txtResult, 1, wxALL, 0);
              // On ajoute la deuxième ligne au wxStaticBoxsizer
              box->Add(line2, 0, wxALL|wxEXPAND, 5);
          // On ajoute le wxStaticBoxsizer au sizer principal
          mainsizer->Add(box, 1, wxALL|wxEXPAND, 5);
      // Et on affecte le sizer principal au wxPanel
      SetSizer(mainsizer);
  }
```

Comme vous pourrez le constater, il n'y a rien de bien particulier dans ce code (je pense que les commentaires devraient suffire à le comprendre).

Vous remarquerez que j'ai pris l'habitude d'indenter le code pour savoir où j'en suis dans la création des contrôles par rapport aux différents sizers. Cette technique n'est bien entendu pas indispensable, mais elle me permet d'éviter bien des erreurs et surtout des oublis.

Nous allons maintenant mettre en place les méthodes événementielles.

- La première sera appelée lors du clic sur le bouton « parcourir » et servira à afficher une boite de dialogue de sélection de fichier. Elle prendra en paramètre un **`wxCommandEvent`** et s'intitulera **`OnButtonBrowseClicked`**.
- La deuxième sera appelée à chaque modification du nom du fichier à analyser (que ce soit automatiquement en passant par la boite de dialogue ci-dessus, ou manuellement si l'utilisateur modifie le nom du fichier lui-même), afin que notre composant soit informé qu'il y a eut un changement. Elle prendra également en paramètre un **`wxCommandEvent`** et s'intitulera **`OnFilenameChanged`**.

Pour ce qui est de la gestion du glisser-déposer, nous verrons cela plus tard car il ne se gère pas de la même façon.

Voici donc dans un premier temps à quoi ressemble le fichier header de notre composant :

```
#ifndef FILE2CHECKPANEL_H_INCLUDED
#define FILE2CHECKPANEL_H_INCLUDED

#include <wx/wx.h>

class File2CheckPanel : public wxPanel
{
    public:
        File2CheckPanel(wxWindow* parent, const wxString& title);
        virtual ~File2CheckPanel();
    private:
        wxTextCtrl *m_txtFileName, *m_txtResult;
        wxButton *m_btnBrowse;
        void CreateAndConnectControls(const wxString& title);
        void OnButtonBrowseClicked(wxCommandEvent& event);
        void OnFilenameChanged(wxCommandEvent& event);
};

#endif // FILE2CHECKPANEL_H_INCLUDED
```

Nous allons maintenant voir le code servant à afficher la boite de dialogue de sélection d'un fichier.

Comme vous vous en doutiez, wxWidgets possède une classe toute prête pour ce type de dialogue. Il s'agit de **wxFileDialog**.

Le premier paramètre du constructeur de cette classe est, comme pour la majorité des classes GUI wxWidgets, le pointeur vers la fenêtre ou le contrôle parent. Nous mettrons le pointeur « **this** » pour indiquer que notre composant est le parent de la boite de dialogue.

Le deuxième paramètre correspond au message affiché dans la barre de titre de la boite de dialogue.

Les troisième et quatrième paramètres correspondent au chemin et au nom de fichier par défaut. Nous les laisserons vides.

Le cinquième paramètre correspond au filtre d'affichage. Comme notre petit utilitaire permet d'analyser n'importe quel fichier, le filtre sera donc « ***.*** ».

Le sixième paramètre est une combinaison de différents styles. Nous utiliserons les styles « **wxFD_OPEN** » pour indiquer qu'il s'agit d'une boite de dialogue d'ouverture de fichier, « **wxFD_FILE_MUST_EXIST** » pour que l'utilisateur ne puisse sélectionner que les fichiers existants (et non entrer un nom de fichier inexistant). Vous pourrez retrouver ces valeurs (ainsi que les autres disponibles) dans la documentation officielle du **wxFileDialog**.

Pour les autres paramètres, nous ne mettrons rien afin d'utiliser les valeurs par défaut.

Afin de faciliter la compréhension du code, je vais dans un premier temps placer les valeurs des différents paramètres dans des variables, et appeler le constructeur avec ces variables, mais il est bien entendu possible de mettre les valeurs directement dans l'appel du constructeur.

Ensuite, après la construction de l'objet **wxFileDialog**, il suffit d'en appeler la méthode « **ShowModal** » pour que la boite de dialogue s'affiche.

Cette méthode donne en retour la valeur **wxID_CANCEL** si l'utilisateur a cliqué sur le bouton « Annuler » de la boite de dialogue (et dans ce cas, aucun fichier n'a été sélectionné), ou **wxID_OK** si l'utilisateur a bien sélectionné un fichier et cliqué sur « Ouvrir ».

Dans le premier cas, nous sortirons purement et simplement de la fonction **OnButtonBrowseClicked**, sans rien faire d'autre.

Dans le deuxième cas, nous pourrons récupérer le chemin complet du fichier sélectionné grâce à la méthode « **GetPath** » de l'objet **wxFileDialog**.

Voici maintenant ce que ça donne réellement :

```
   void File2CheckPanel::OnButtonBrowseClicked(wxCommandEvent& event)
   {
       // Les variables contenant les paramètres du constructeur de l'objet
wxFileDialog
       wxString sMsg=_T("Sélectionnez un fichier");
       wxString sDefPath=_T("");
       wxString sDefName=_T("");
       wxString sFilter=_T("Tous les fichiers|*.*");
       int iStyle=wxFD_OPEN | wxFD_FILE_MUST_EXIST;
       // On crée l'objet wxFileDialog
       wxFileDialog fdlg(this, sMsg, sDefPath, sDefName, sFilter, iStyle);
       // On l'affiche, et si la valeur de retour est wxID_CANCEL, on sort
       if (fdlg.ShowModal()==wxID_CANCEL) return;
       // On récupère le chemin complet du fichier sélectionné
       // Et on le place dans la zone de texte prévue à cet effet
       m_txtFileName->SetValue(fdlg.GetPath());
   }
```

Voilà, c'est aussi simple que ça.

Voyons maintenant le code de la deuxième méthode événementielle de notre composant.

Il s'agit de la fonction qui sera appelée à chaque changement dans le texte correspondant au nom du fichier à analyser. Elle sera donc appelée lorsque l'utilisateur entre manuellement le nom du fichier, mais également lorsque cette valeur est modifiée à la fin de la fonction ci-dessus.

Comme l'utilisateur peut modifier cette valeur manuellement, elle peut donc ne pas correspondre à un fichier existant. Il faut donc faire cette vérification (wxWidgets nous propose pour cela une fonction **`wxFileExists`**).

Pour le reste du code de cette fonction, il s'agit simplement d'une adaptation du code contenu dans le fichier « md5.c » dont je vous ait parlé dans le premier chapitre de ce cours. La seule différence réside dans l'utilisation de la classe wxWidgets « **`wxFile`** » pour la lecture du fichier à analyser.

Voici donc le contenu de la fonction « **`OnFilenameChanged`** » :

```
   void File2CheckPanel::OnFilenameChanged(wxCommandEvent& event)
   {
       // On récupère le chemin complet du fichier à analyser
       wxString sFName=m_txtFileName->GetValue() ;
       // On supprime les éventuels espaces de début et de fin
       sFName.Trim(false);
       sFName.Trim(true) ;

       // On vérifie que le fichier à analyser est bien valide
       // Sinon, on efface la zone de texte du résultat et on sort
       if ((sFName.IsEmpty()) || (!wxFileExists(sFName)))
       {
           m_txtResult->Clear();
           return;
       }

       // Les variables pour le calcul de la somme MD5
       int len;
       md5_byte_t buffer[512];
       md5_state_t state;
       md5_byte_t digest[16];

       // On essaye d'ouvrir le fichier à analyser en lecture.
       wxFile file(sFName, wxFile::read);
       // Si l'ouverture échoue, on efface la zone de texte du résultat et on sort
       if (!file.IsOpened())
       {
```

```
            m_txtResult->Clear();
            return;
        }

        // On démarre le calcul de la somme MD5
        md5_init(&state);
        do
        {
            len=file.Read(buffer, 512);
            md5_append(&state,(const md5_byte_t *)buffer,len);
        } while(!file.Eof());
        md5_finish(&state, digest);
        file.Close();

        // On récupère le résultat dans une variable wxString
        wxString sRes=_T("");
        for( int i=0;i<16;i++) sRes.Append(wxString::Format(_T("%02X"),digest[i]));
        // On affiche le résultat dans la zone de texte prevue à cet effet
        m_txtResult->SetValue(sRes);
    }
```

Pour que le code ci-dessus puisse être compilé il faut penser à inclure le header « **md5.h** » ainsi que celui de la classe « **wxFile** » au début du fichier « **File2CheckPanel.cpp** » :

```
#include "File2CheckPanel.h"
#include "md5.h"
#include <wx/file.h>
```

Maintenant que les deux fonctions événementielles sont codées, il ne reste plus qu'à les connecter au gestionnaire d'événements de notre composant personnalisé.

Pour cela, nous n'allons pas utiliser la méthode « classique » des tables d'événements, mais une autre façon de faire appelée « connexion dynamique ».

Je ne sais pas si vous avez remarqué, mais il n'a pour l'instant jamais été question de définir les identifiants des contrôles utilisés. Dans le code de création des contrôles, j'ai volontairement remplacé les IDs par la valeur -1, pour indiquer aux libs wxWidgets de se débrouiller pour la création et l'affectation de ces identifiants.

Si vous jetez un coup d'œil à la documentation officielle, vous vous apercevrez que tous les contrôles sont dérivés de la classe **wxEvtHandler** qui est la classe servant à gérer les messages reçus.

Cette classe (et du coup nos contrôles) possède une méthode « **Connect** » qui sert justement à la connexion dynamique, et l'une des différentes formes de cette méthode n'utilise justement pas l'identifiant du contrôle concerné (notez qu'il aurait été facile de le retrouver avec la fonction « **GetId** » de la classe **wxWindow** dont dérivent également tous les contrôles).

Les deux paramètres obligatoires de la méthode **Connect** sont le type de l'événement et la fonction devant recevoir l'événement lui-même. Et d'après la documentation, cette dernière, doit obligatoirement être convertie au type de fonction correspondant au type d'événement utilisé. Par exemple, pour un **wxCommandEvent** (utilisé pour nos deux méthodes événementielles), il faut la convertir au type **wxCommandEventHandler** ; si nous avions utilisé un **wxCloseEvent**, il aurait fallu la convertir au type **wxCloseEventHandler**.

Il y a ensuite un paramètre nommé **userData** de type **wxObject*** dont la valeur par défaut est nulle, et qui sert à passer une valeur spécifique par l'intermédiaire de l'événement.

Quand au dernier paramètre, nommé **eventSink**, il spécifie l'objet **wxEvtHandler*** dont fait partie la fonction appelée. Si cette valeur est nulle (ce qui est la valeur par défaut), le pointeur « **this** » est utilisé.

Dans notre cas, pour connecter le bouton à la fonction **OnButtonBrowseClicked**, les paramètres seront donc :

- Type de l'événement : **wxEVT_COMMAND_BUTTON_CLICKED** (pour ceux qui se demandent comment trouver cette valeur, voir la documentation officielle de la classe **wxButton**).
- Fonction : **File2CheckPanel::OnButtonBrowseClicked** (à convertir en **wxCommandEventHandler**)
- **userData** : **NULL**
- **eventSink** : **this** (la fonction **OnButtonBrowseClicked** fait partie de la classe **File2CheckPanel** dans laquelle nous placerons la commande de connexion).

Ce qui donnera :

```
m_btnBrowse->Connect(wxEVT_COMMAND_BUTTON_CLICKED,
    wxCommandEventHandler(File2CheckPanel::OnButtonBrowseClicked), NULL, this);
```

Pour connecter la deuxième méthode événementielle à la zone de texte contenant le chemin complet du fichier à analyser, les paramètres sont les suivants :

- Type de l'événement : **wxEVT_COMMAND_TEXT_UPDATED**
- Fonction : **File2CheckPanel::OnFilenameChanged** (à convertir en **wxCommandEventHandler**)
- **userData** : **NULL**
- **eventSink** : **this**

La ligne de code est donc :

```
m_txtFileName->Connect(wxEVT_COMMAND_TEXT_UPDATED,
    wxCommandEventHandler(File2CheckPanel::OnFilenameChanged), NULL, this);
```

Vous l'aurez compris, les deux lignes de code ci-dessus sont à insérer à la fin de la méthode « **CreateAndConnectControls** ».

Voilà, notre composant autonome est maintenant terminé dans sa version de base. Nous verrons plus tard pour y ajouter la fonctionnalité de glisser/déposer.

Pour l'heure, il est temps de placer notre composant dans une fenêtre afin de vérifier qu'il fonctionne bien.

4) La classe d'application et la fenêtre principale

Travaillant actuellement sous Code::Blocks, j'ai pour ma part créé un projet classique wxWidgets nommé **wxMD5Check**.

Je me retrouve donc avec une classe dérivée de **wxApp** nommée **wxMD5CheckApp** (fichiers **wxMD5CheckApp.h** et **wxMD5CheckApp.cpp**), et une fenêtre principale dérivée de **wxFrame** nommée **wxMD5CheckFrame** (fichiers **wxMD5CheckMain.h** et **wxMD5checkMain.cpp**).

La fenêtre principale est créée depuis la fonction « **OnInit** » de la classe d'application.

Elle possède une variable membre privée de type **wxFile2CheckPanel*** nommée **m_panel1** (qui pour l'instant n'est pas forcément nécessaire, mais qui nous servira plus tard lorsque nous aurons placé les deux panels et que nous voudrons comparer deux sommes MD5).

Elle possède également une méthode « **OnClose** » (créée automatiquement par l'assistant nouveau projet de Code::Blocks) servant à détruire la fenêtre lorsque l'utilisateur clique sur la croix de fermeture en hau à droite. Cette méthode est connectée à son événement par la méthode classique des tables d'événements.

Vous constaterez également que je n'ai pas laissé le style par défaut de la fenêtre, et que j'en ai mis un permettant d'obtenir une fenêtre non-redimensionnable.

J'ai de plus utilisé un fichier xpm (que vous pourrez retrouver dans le dossier « art » des libs et contenant l'icône au format xpm de façon à avoir le même code pour Windows et Linux (ce dernier ne prenant pas en charge les icônes dans les ressources).

Dernière précision : la classe de fenêtre possède une méthode « **CreateAndConnectControls** » dont vous aurez compris l'utilité.

Le reste du code n'ayant rien de particulier, je vous le livre dans son intégralité :

Fichier « wxMD5CheckApp.h » :

```
#ifndef WXMD5CHECKAPP_H_INCLUDED
#define WXMD5CHECKAPP_H_INCLUDED

#include <wx/wx.h>

class wxMD5CheckApp : public wxApp
{
    public:
        virtual bool OnInit();
};

#endif // WXMD5CHECKAPP_H_INCLUDED
```

Fichier « wxMD5CheckApp.cpp » :

```
#include "wxMD5CheckApp.h"
#include "wxMD5CheckMain.h"

IMPLEMENT_APP(wxMD5CheckApp);

bool wxMD5CheckApp::OnInit()
{
    wxMD5CheckFrame* frame = new wxMD5CheckFrame(NULL, _T("wxMD5Check"));

    frame->Show();

    return true;
}
```

Fichier « wxMD5CheckMain.h » :

```
#ifndef WXMD5CHECKMAIN_H_INCLUDED
#define WXMD5CHECKMAIN_H_INCLUDED

#include <wx/wx.h>

class File2CheckPanel;

class wxMD5CheckFrame: public wxFrame
{
    public:
        wxMD5CheckFrame(wxFrame *frame, const wxString& title);
        ~wxMD5CheckFrame();
    private:
        void CreateAndConnectControls();
        void OnClose(wxCloseEvent& event);
        File2CheckPanel* m_panel1;
        DECLARE_EVENT_TABLE()
};

#endif // WXMD5CHECKMAIN_H_INCLUDED
```

Fichier « wxMD5CheckMain.cpp » :

```
#include "wxMD5CheckMain.h"
```

```
#include "File2CheckPanel.h"

#include "wxwin32x32.xpm"

BEGIN_EVENT_TABLE(wxMD5CheckFrame, wxFrame)
    EVT_CLOSE(wxMD5CheckFrame::OnClose)
END_EVENT_TABLE()

wxMD5CheckFrame::wxMD5CheckFrame(wxFrame *frame, const wxString& title)
    : wxFrame(frame, -1, title, wxDefaultPosition, wxDefaultSize, wxMINIMIZE_BOX|
wxSYSTEM_MENU|wxCAPTION|wxCLOSE_BOX)
{
    CreateStatusBar();
    SetStatusText(_T("Hello wxWidgets user!"));

    CreateAndConnectControls();

    SetIcon(wxwin32x32_xpm);

}

wxMD5CheckFrame::~wxMD5CheckFrame()
{
}

void wxMD5CheckFrame::CreateAndConnectControls()
{
    // Création du sizer principal
    wxBoxSizer* mainsizer=new wxBoxSizer(wxVERTICAL);
        // Création du premier panel autonome
        m_panel1=new File2CheckPanel(this, _T("Fichier à analyser : "));
        mainsizer->Add(m_panel1, 0, wxALL|wxEXPAND, 0);
    // On applique le sizer à la fenêtre
    SetSizer(mainsizer);
    // Et on lui demande de mettre à jour la taille de la fenêtre
    // en fonction des contrôles qu'elle contient
    mainsizer->SetSizeHints(this);
}

void wxMD5CheckFrame::OnClose(wxCloseEvent &event)
{
    Destroy();
}
```

Voilà, c'est terminé.

Vous devriez normalement obtenir une petite application qui fonctionne (essayez de cliquer sur le bouton « parcourir » ou d'entrer un nom de fichier manuellement).

Vous voulez ajouter un deuxième panel autonome afin d'obtenir la même chose que sur les captures d'écran du début ? Pas de problème !

Il vous suffit d'ajouter une deuxième variable **`m_panel2`** à la classe **`wxMD5CheckFrame`** :

```
#ifndef WXMD5CHECKMAIN_H_INCLUDED
#define WXMD5CHECKMAIN_H_INCLUDED

#include <wx/wx.h>

class File2CheckPanel;

class wxMD5CheckFrame: public wxFrame
{
```

```
    private:
        File2CheckPanel* m_panel1,* m_panel2;
    ............
};

#endif // WXMD5CHECKMAIN_H_INCLUDED
```

Et maintenant ajoutez le code correspondant à la création de ce deuxième panel :

```
void wxMD5CheckFrame::CreateAndConnectControls()
{
    // Création du sizer principal
    wxBoxSizer* mainsizer=new wxBoxSizer(wxVERTICAL);
        // Création du premier panel autonome
        m_panel1=new File2CheckPanel(this, _T("Fichier N°1 : "));
        mainsizer->Add(m_panel1, 0, wxALL|wxEXPAND, 0);
        // Création du deuxième panel autonome
        m_panel2=new File2CheckPanel(this, _T("Fichier N°2 : "));
        mainsizer->Add(m_panel2, 0, wxALL|wxEXPAND, 0);
    // On applique le sizer à la fenêtre
    SetSizer(mainsizer);
    // Et on lui demande de mettre à jour la taille de la fenêtre
    // en fonction des contrôles qu'elle contient
    mainsizer->SetSizeHints(this);
}
```

Et le tour est joué.

5) Accepter le glisser/déposer de fichiers

Comme le titre de ce chapitre l'indique, nous allons maintenant permettre au panel autonome de recevoir le fichier à analyser par glisser/déposer.

Mais avant d'attaquer le code, voyons un peu de théorie : voici la marche à suivre pour permettre à une application d'être une cible « drag'n'drop ».

Il faut en fait que le contrôle devant recevoir quelque chose par glisser/déposer ait un objet **`wxDropTarget`** associé (par l'intermédiaire de la fonction **`wxWindow::SetDropTarget`**). Il faut créer une classe dérivée de **`wxDropTarget`** (qui est en fait une classe abstraite) et surcharger ses méthodes virtuelles pures afin de gérer les données transférées et accepter ou refuser le drag'n'drop.

Dans notre cas, nous n'allons pas tout à fait suivre ces indications, car wxWidgets a déjà prévu une classe pour le transfert de fichiers : **`wxFileDropTarget`**. De cette façon, nous n'aurons qu'une méthode virtuelle à surcharger : la méthode **`OnDropFiles`**.

Nous pourrons passer en paramètre au constructeur de la classe le pointeur vers la zone de texte devant recevoir le nom du fichier. De cette façon, le chemin du fichier sera automatiquement mis à jour.

Pour revenir sur la méthode à surcharger, elle possède trois paramètres :

- La coordonnée X du curseur au moment où l'utilisateur relâche la souris.
- La coordonnée Y du curseur au moment où l'utilisateur relâche la souris.
- Un tableau **`wxArrayString`** contenant la liste des fichiers transférés

Vous aurez compris que seul de dernier paramètre nous intéresse, et plus précisément la première entrée de ce tableau. En effet, notre panel autonome est capable d'afficher la somme MD5 d'un seul fichier.

Vous allez donc voir que le code de notre classe dérivée de **`wxFileDropTarget`** est grandement simplifié :

Fichier « MyFileDropTarget.h »

```
#ifndef MYFILEDROPTARGET_H_INCLUDED
#define MYFILEDROPTARGET_H_INCLUDED

#include <wx/wx.h>
#include <wx/dnd.h>

class MyFileDropTarget : public wxFileDropTarget
{
   public:
         MyFileDropTarget(wxTextCtrl* target);
         virtual ~MyFileDropTarget();
         virtual bool OnDropFiles(wxCoord x, wxCoord y, const wxArrayString
&filenames);
    private:
         wxTextCtrl *m_txtTarget;
};

#endif // MYFILEDROPTARGET_H_INCLUDED
```

Fichier « MyFileDropTarget.cpp »

```
#include "MyFileDropTarget.h"

MyFileDropTarget::MyFileDropTarget(wxTextCtrl *target)
{
   m_txtTarget=target;
}

MyFileDropTarget::~MyFileDropTarget()
{
   // dtor
}

bool MyFileDropTarget::OnDropFiles(wxCoord x, wxCoord y, const wxArrayString
&filenames)
{
   // On ne peut accepter qu'un seul fichier à la fois
   // On ne prend donc que le premier élément du wxArrayString
   // et on le met dans le wxTextCtrl
   m_txtTarget->SetValue(filenames[0]);
   return true;
}
```

Il ne reste plus qu'à associer une instance de cette classe au contrôle devant recevoir le fichier (dans notre cas, le **`wxTextCtrl`** contenant le chemin du fichier à analyser.

Il faut bien entendu penser à inclure le fichier en-tête de cette nouvelle classe dans le fichier source du panel autonome :

```
#include "File2CheckPanel.h"
#include "md5.h"
#include <wx/file.h>
#include "MyFileDropTarget.h"
```

Et maintenant, associons un nouveau « **`MyFileDropTarget`** » au **`wxTextCtrl`** après sa création :

```
void File2CheckPanel::CreateAndConnectControls(const wxString& title)
{
    .........

    // Connexion des contrôles aux méthodes événementielles
    m_btnBrowse->Connect(wxEVT_COMMAND_BUTTON_CLICKED,
wxCommandEventHandler(File2CheckPanel::OnButtonBrowseClicked), NULL, this);
```

```
      m_txtFileName->Connect(wxEVT_COMMAND_TEXT_UPDATED,
wxCommandEventHandler(File2CheckPanel::OnFilenameChanged), NULL, this);
      // Association d'un MyFileDropTarget au wxTextCtrl du chemin du fichier à
analyser
      m_txtFileName->SetDropTarget(new MyFileDropTarget(m_txtFileName));
  }
```

Vous pouvez tester : il est désormais possible de faire glisser un fichier dans une des deux zones de texte recevant les noms des fichiers à analyser.

6) Demander à la fenêtre principale de comparer les sommes MD5

A ce stade, notre application est entièrement fonctionnelle.

Il reste encore une petite chose à ajouter : nous avons placé deux instances de notre panel autonome dans la fenêtre principale afin de pouvoir comparer les sommes MD5 de deux fichiers.

Il faut donc que la fenêtre soit informée du fait qu'un fichier vient d'être analysé, et qu'elle doit maintenant comparer sa somme MD5 avec celle d'un éventuel autre fichier.

Pour que la comparaison se fasse, nous avons trois possibilités :

- Ajouter un bouton afin que l'utilisateur demande manuellement la comparaison.
- Ajouter un timer qui, par exemple toutes les secondes, va vérifier s'il y a deux fichier à comparer
- Poster un événement depuis le panel autonome pour indiquer que la somme MD5 d'un fichier vient d'être calculée.

Vous l'aurez compris : nous allons nous occuper de la troisième solution qui est à mon goût la plus propre.

Le fonctionnement va être le suivant :

- Le panel autonome calcule la somme MD5 d'un fichier lorsque le contenu de la zone de texte est modifié.
- Il poste un événement personnalisé indiquant que la somme MD5 a été actualisée
- La fenêtre principale reçoit cet événement, demande la somme MD5 des deux panels, les compare, et affiche le résultat (sommes égales ou différentes) dans la barre d'état.

Nous allons donc dans un premier temps ajouter une fonction permettant à la fenêtre de récupérer la somme MD5 calculée par un panel autonome : cette fonction devra tout simplement lire le contenu de la zone de texte « résultat » et le retourner sous la forme d'un **`wxString`**.

Voici donc la partie à ajouter dans le header de la classe « File2CheckPanel » :

```
class File2CheckPanel : public wxPanel
{
    public:
        .........
        wxString GetMD5Value() const;
    private:
        .........
};
```

Et voici le code source de la nouvelle fonction :

```
wxString File2CheckPanel::GetMD5Value() const
{
    return m_txtResult->GetValue();
}
```

Nous allons maintenant créer un nouveau type d'événement personnalisé. Pour cela, wxWidgets nous fourni une fonction permettant de créer un nouvel identifiant de type d'événement en utilisant une valeur qui n'est pas encore utilisée : **`wxNewEventType()`**

Comme on voudrait que cet événement soit utilisable depuis chaque classe qui instancie le panel autonome, nous mettrons la déclaration de l'événement dans le header du panel, et sa définition dans le code source de cette même classe :

Pour la déclaration :

```
extern const wxEventType wxEVT_CHECKSUM_CHANGED;
```

Et pour la définition :

```
const wxEventType wxEVT_CHECKSUM_CHANGED = wxNewEventType();
```

Nous pouvons désormais modifier le code du panel autonome afin que ce nouvel événement soit posté quand la somme MD5 est modifiée.

Le plus simple étant de poster l'événement à chaque fois que la valeur de la zone de texte « résultat » est modifiée, nous allons ajouter une fonction événementielle au panel autonome, sur le même principe que pour la modification du chemin du fichier à analyser. Cette fonction va uniquement créer un objet **wxCommandEvent** de type **wxEVT_CHECKSUM_CHANGED** et la placer dans la queue des événements du panel. Comme les **wxCommandEvent** se propagent s'ils ne sont pas traités, la fenêtre le recevra et pourra agir en conséquence.

Tout d'abord, ajouter la déclaration de cette nouvelle fonction dans le header du panel autonome :

```
class File2CheckPanel : public wxPanel
{
    .........
    private:
        .........
        void OnResultChanged(wxCommandEvent& event);
};
```

Ensuite la définition de cette même fonction dans le fichier source du panel :

```
void File2CheckPanel::OnResultChanged(wxCommandEvent& event)
{
    wxCommandEvent evt(wxEVT_CHECKSUM_CHANGED, GetId());
    AddPendingEvent(evt);
}
```

Et on ajoute la commande permettant de connecter cette fonction à la table des événements :

```
void File2CheckPanel::CreateAndConnectControls(const wxString& title)
{
    .........
    // Connexion des contrôles aux méthodes événementielles
    m_btnBrowse->Connect(wxEVT_COMMAND_BUTTON_CLICKED,
wxCommandEventHandler(File2CheckPanel::OnButtonBrowseClicked), NULL, this);
    m_txtFileName->Connect(wxEVT_COMMAND_TEXT_UPDATED,
wxCommandEventHandler(File2CheckPanel::OnFilenameChanged), NULL, this);
    m_txtResult->Connect(wxEVT_COMMAND_TEXT_UPDATED,
wxCommandEventHandler(File2CheckPanel::OnResultChanged), NULL, this);
    // Association d'un MyFileDropTarget au wxTextCtrl du chemin du fichier à
analyser
    m_txtFileName->SetDropTarget(new MyFileDropTarget(m_txtFileName));
}
```

Il ne reste maintenant plus qu'à gérer cela du côté de la fenêtre principale en ajoutant une fonction événementielle (**OnChecksumChanged**) et en la connectant à la table d'événements.

Fichier en-tête de la fenêtre principale :

```
class wxMD5CheckFrame: public wxFrame
{
    .........
    private:
        .........
        void OnChecksumChanged(wxCommandEvent& event);
        .........
};
```

Fichier source de la fenêtre principale :

```
void wxMD5CheckFrame::OnChecksumChanged(wxCommandEvent &event)
{
    // On récupère la somme MD5 du premier fichier
    wxString sVal1=m_panel1->GetMD5Value();
    // On récupère la somme MD5 du deuxième fichier
    wxString sVal2=m_panel2->GetMD5Value();
    // La variable pour le message final à afficher
    wxString sTxt=_T("");
    // On ne fait la comparaison que si les deux fichiers sont valides
    if ((!sVal1.IsEmpty())&&(!sVal2.IsEmpty()))
    {
        sTxt=_T("Les sommes MD5 des deux fichiers sont ");
        sTxt << ((sVal1==sVal2)?_T("identiques."):_T("différentes."));
    }
    // On affiche le résultat dans la barre d'état de la fenêtre
    SetStatusText(sTxt);
}
```

On n'oublie pas d'ajouter la connexion dynamique de cette fonction à la table d'événements de la fenêtre :

```
void wxMD5CheckFrame::CreateAndConnectControls()
{
    .........
    // On connecte tous les événements wxEVT_CHECKSUM_CHANGED
    Connect(wxEVT_CHECKSUM_CHANGED,
wxCommandEventHandler(wxMD5CheckFrame::OnChecksumChanged), NULL, this);
}
```

7) Conclusion

Et voilà, c'est terminé.

Nous disposons maintenant d'une application entièrement fonctionnelle.

Vous pourrez retrouver le code source complet dans la rubrique « projets » de www.wxdev.fr.

N'hésitez pas à laisser vos commentaires et vos suggestions d'amélioration avec de faire évoluer ce tutoriel.

En attendant, bonne prog à tous, et à bientôt sur wxDev.fr

Xaviou