

Réalisation d'une application avec wxWidgets

Fred Cailleau-Lepetit



Paternité - Pas d'Utilisation Commerciale - Pas de Modification 2.0 France

Vous êtes libres :

- de reproduire, distribuer et communiquer cette création au public

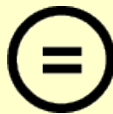
Selon les conditions suivantes :



Paternité. Vous devez citer le nom de l'auteur original.



Pas d'Utilisation Commerciale. Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.



Pas de Modification. Vous n'avez pas le droit de modifier, de transformer ou d'adapter cette création.

- A chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les conditions contractuelles de mise à disposition de cette création.
- Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.

Ce qui précède n'affecte en rien vos droits en tant qu'utilisateur (exceptions au droit d'auteur : copies réservées à l'usage privé du copiste, courtes citations, parodie...)

Ceci est le Résumé Explicatif du [Code Juridique \(la version intégrale du contrat\)](#).

Table des matières

1	Présentation.....	1
	But.....	1
	Convention typographique.....	2
	Outils utilisés.....	2
	Éditeur.....	2
	Bibliothèques.....	2
	Utilitaires divers.....	3
2	Installation du poste de développement.....	5
	Installation de Dev-Cpp.....	5
	Installation des devpacks wxWidgets.....	5
	Mise à jour de Dev-C++.....	7
3	Début du projet.....	9
	La classe application.....	11
	La classe de fenêtre principale de votre application.....	12
4	Le menu principal de votre application.....	15
	Menu File.....	16
	Menu Local.....	17
	Menu Remote.....	17
	Menu Help.....	18
	Finition du menu.....	18
5	La barre d'outils.....	21
	La classe wxToolBar.....	22
6	Création des contrôles de l'application.....	27
	Les classes wxSizer.....	27
	La classe wxBoxSizer.....	28
	La classe wxStaticBoxSizer.....	28
	La classe wxGridSizer.....	28
	La classe wxFlexGridSizer.....	29
	La classe wxGridBagSizer.....	29
	Utilisation des wxSizer dans l'application.....	29
7	Une autre classe de fenêtre : wxDialog.....	35

Différence principale avec entre wxDialog et wxFrame.....	35
Création d'une classe dérivée de wxDialog.....	35
Création des contrôles du dialogue.....	36
Contrôle et validation des données.....	38
Les classes wxValidator.....	39
La fonction Validate.....	40
8 La gestion des événements.....	43
Qu'est ce qu'un événement?.....	43
La table des événements.....	43
Les événements menu.....	44
Les événements contrôles.....	47
Les événements de wxTextCtrl.....	47
Les événements de wxSplitterWindow.....	48
Les événements de wxListCtrl.....	48
Affichage de la liste locale.....	50
Implémentation de OnMnuChangeLocalClick.....	53
Implémentation de OnMnuCreateLocalClick.....	53
Implémentation de OnMnuDelLocalClick.....	53
Implémentation de OnMnuRefreshLocalClick.....	55
Implémentation de OnLocalFilesItemActivated.....	55
Implémentation de OnFilesChangeSelection.....	55
9 Les sockets avec wxWidgets.....	59
Qu'est ce qu'un socket?.....	59
Les classes dérivées de wxSocketBase.....	59
La classe wxSocketServer.....	60
La Classe wxSocketClient.....	60
La classe wxProtocol.....	61
La classe wxHTTP.....	61
La classe wxFTP.....	61
Les événements socket.....	63
Création d'une classe dérivée de wxFTP.....	66
10 Le multi-tâches.....	73
Qu'est ce que le multi-tâches?.....	73
Les classes pour la gestion du multi-tâches.....	73
La classe wxThread.....	74
La classe wxCriticalSection.....	74
La classe wxCriticalSectionLocker.....	74
Création de classes dérivées de wxThread.....	74
La classe wxBaseThread.....	77
La classe wxManageRemoteThread.....	80
La classe wxTransThread.....	86
La classe wxDownloadThread.....	90
La classe wxUploadThread.....	92
Intégration des classes de tâches.....	94
Intégration de la tâche wxManageRemoteThread.....	94
Intégration des tâches de transfert.....	98

11 Le glisser/déposer (Drag and drop).....	105
Qu'est ce que le glisser/déposer?.....	105
Les classes pour la gestion du glisser/déposer.....	105
La classe wxDataObjectSimple.....	106
La classe wxDropTarget.....	106
Création de classe pour le glisser/déposer.....	107
La classe wxDndUploadFile.....	108
La classe wxDndUploadFileDataObject.....	108
La classe wxDndRemoteTarget.....	110
La classe wxDndDownloadFile.....	111
La classe wxDndDownloadFileDataObject.....	112
La classe wxDndLocaleTarget.....	114
Intégration des classes de glisser/déposer.....	115
 12 Internationalisation.....	 117
Utilisation de poEdit.....	117
La classe wxLocale.....	122
Intégration de l'internationalisation.....	122
 13 Annexe A : Index des illustrations.....	 125
 14 Annexe B : Remerciements.....	 127

1 Présentation

But

Au cours de ce tutoriel le lecteur sera amené à construire une application complète. En passant par toutes les étapes de réalisation, il abordera la plupart des concept de la bibliothèque wxWidgets. Le développement se fera sous Microsoft Windows™ en utilisant l'environnement de développement Dev-Cpp. L'application servant de base à cet ouvrage est un client FTP dont vous pouvez voir l'interface sur l'image ci-dessous.

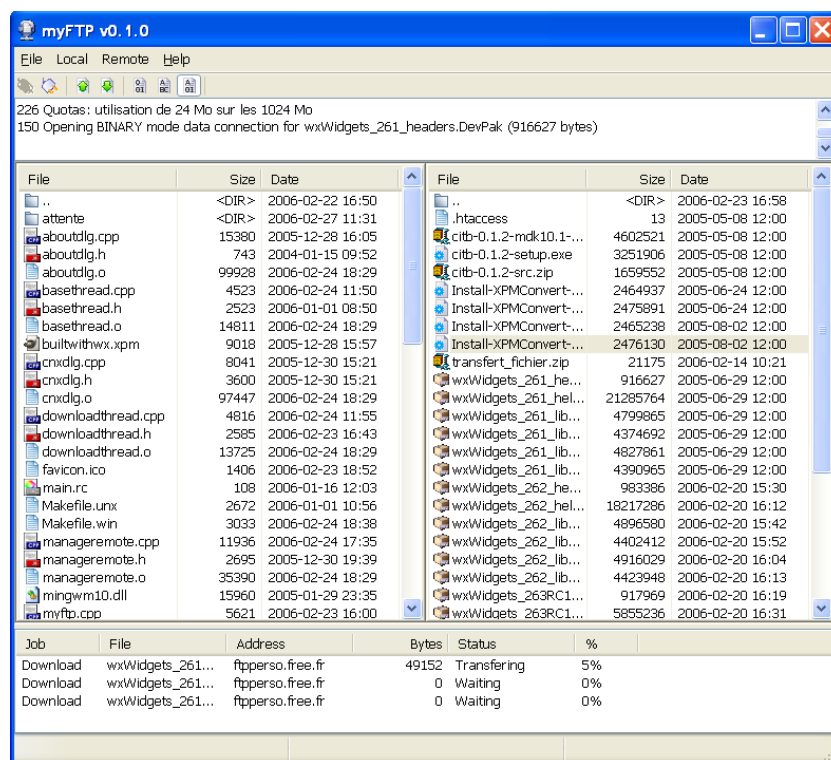


Illustration 1: MyFTP 0.1.0 (Windows XP)

Convention typographique

Les conventions typographiques suivantes ont été adoptées :

Italique

Noms et extensions de fichiers, fichiers exécutables, commandes et mise en relief d'un mot ou d'un groupe de mots.

Italique souligné bleu

URL et adresse Internet.

Police à espacement fixe

Code source C/C++ et autres commandes et instructions figurant dans le texte, informations devant être saisies sans modifications sur la ligne de commande.

↵

Dans le code source et autres commandes, indique un retour à la ligne involontaire dû au contraintes de la mise en page.

Gras

Menu, Onglet, bouton.

[Gras]

Touche ou combinaison de touche

Notes

Lorsque le contexte le justifie, des notes et des conseils sont affichés de la manière suivante :



L'ampoule avec un point d'exclamation signale une note ou un conseil, en marge du texte principal.

Outils utilisés

Éditeur

Comme il est indiqué au paragraphe 1.1, l'environnement de développement utilisé sera Dev-Cpp. Cet éditeur est un logiciel libre sous licence GPL. Vous pouvez le télécharger à l'adresse suivante : <http://www.bloodshed.net/dev/devcpp.html>.



Si vous ne possédez pas GCC, pensez bien à télécharger la version de Dev-Cpp avec GCC.

Bibliothèques

Les sources de la bibliothèque wxWidgets sont disponibles sur le site <http://www.wxwidgets.org/>. Bien sur si vous téléchargez les sources, il vous faudra

compiler la bibliothèque. Heureusement vous trouverez sur mon site des paquets pour Dev-Cpp de wxWidgets à l'URL suivante : <http://cfred.free.fr/download.php#wxdevpak>.

Utilitaires divers

Utilitaire de conversion d'image : XPM Convert
(<http://cfred.free.fr/xpmconvert.php>).

Éditeur de dialogue wxWidgets : DialogBlocks en version d'essai
(<http://www.anthemion.co.uk/dialogblocks/>).

Outils de comparaison de fichiers : Winmerge (<http://winmerge.org/>).

Éditeur de catalogue pour l'internationalisation : poEdit (<http://www.poedit.org/>).

2

Installation du poste de développement

Installation de Dev-Cpp

Après avoir téléchargé Dev-Cpp à l'URL indiquée au paragraphe 1.3.2, vous devez maintenant installer ce logiciel. Double cliquez sur le fichier devcpp-4.9.9.2_setup.exe qui se trouve à l'endroit où vous avez sauvegardé ce fichier lors de son téléchargement.

Suivez les instructions d'installation affichées à l'écran. Choisissez l'installation « Full ». Par défaut le chemin d'installation est *C:\Dev-Cpp*. Si vous modifiez ce chemin vous devrez être très attentif lors de l'installation des devpacks wxWidgets et ne pas oublier de modifier les « templates » de projets wxWidgets.



Attention le nom du fichier exécutable peut varier en fonction de la version disponible. Celle indiquée ci-dessus est la 4.9.9.2, il est fort possible que d'autre version existe au moment où vous effectuerez le téléchargement.

Installation des devpacks wxWidgets

Sur le site de téléchargement vous trouverez six devpacks de wxWidgets, vous pouvez bien sûr récupérer les six et les installer. Mais dans cet ouvrage vous n'aurez besoin que des deux devpacks suivant :

- Fichiers entêtes pour wxWidgets
- Bibliothèque Ansi statique pour wxWidgets

Une fois ces deux fichiers téléchargés et sauvegardés sur votre disque dur, double cliquez sur le devpack des entêtes pour wxWidgets. Cliquez sur le bouton **Next** tout en suivant les instructions à l'écran. Une fois l'installation terminée cliquez sur le bouton **Finish**. La fenêtre du Dev-C++ Package Manager s'affiche, fermez cette fenêtre.

Maintenant double cliquez sur le devpack de la bibliothèque Ansi statique. Cliquez sur le bouton **Next** en suivant les instructions affichées à l'écran. Attention sur le deuxième écran des informations importantes sont affichées si vous n'avez pas

installé Dev-Cpp dans le répertoire par défaut *C:\Dev-Cpp*.



Dans le cas où vous n'avez pas installé Dev-Cpp dans le répertoire par défaut, vous devrez éditer le fichier *wxXXXAS Application.template* (XXX est le numéro de version de wxWidgets) qui se trouve dans le sous-répertoire *Templates* du répertoire d'installation de Dev-Cpp. Vous devrez modifier les lignes commençant par *Libs=*, *Includes=* et *ResourceIncludes=* afin que celle-ci correspondent à votre répertoire d'installation.

En plus des deux devpacks indiqués, je vous conseille vivement de télécharger et d'installer « Aide et exemples wxWidgets » : celui-ci contient un fichier d'aide au format Windows pour wxWidgets ainsi qu'une multitude d'exemple d'utilisation des classes fournies dans wxWidgets.

Mise à jour de Dev-C++

Pour effectuer une mise à jour choisissez dans le menu **Outils** l'item de menu **Nouvelles versions/packages**.

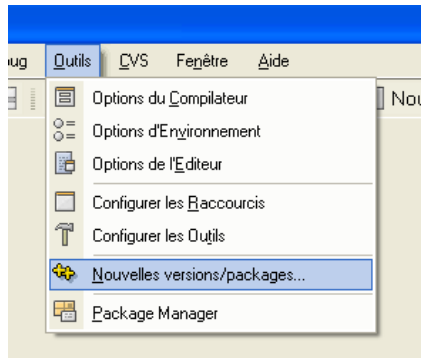


Illustration 2: Mise à jour Dev-C++

Une boîte de dialogue de mise à jour s'affiche, cliquez sur le bouton **Check for updates**, la liste se remplit de mises à jour disponibles. Choisissez les mises à jour suivantes :

- Windows 32 API
- MinGW Runtime
- GNU Debugger

Puis cliquez sur le bouton **Download selected**. Une fois les devpacks téléchargés, l'installation commence, suivez les instructions affichées à l'écran. Pour terminer fermez le dialogue de mise à jour en cliquant sur le bouton **Close**.

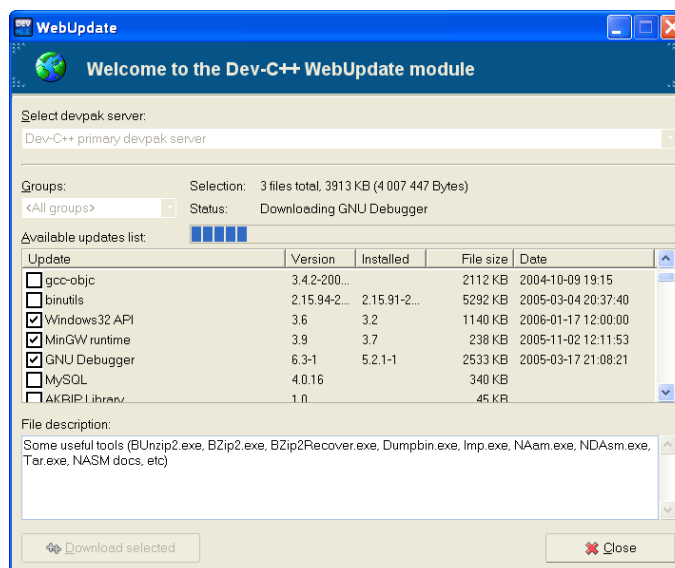


Illustration 3: Dialogue de mise à jour de Dev-C++

3 Début du projet

Pour lancer Dev-Cpp, ouvrez le menu **démarrer**, **Tous les programmes** puis **Bloodshed Dev-C++** et enfin **Dev-C++**.

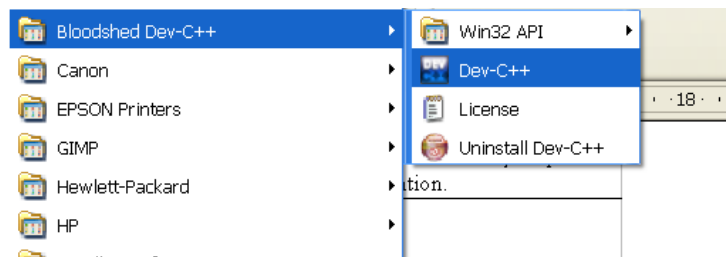


Illustration 4: Lancement de Dev-C++

Vous devez à présent créer un nouveau projet. Dans le menu **Fichier** de Dev-C++ choisissez **Nouveau** puis **projet...**

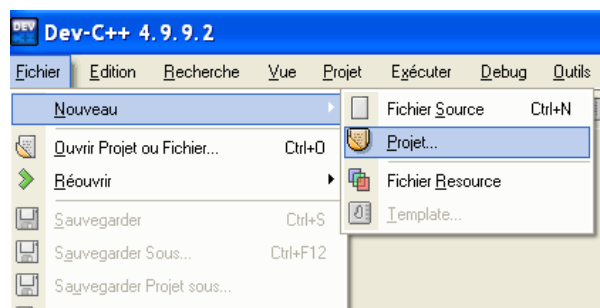


Illustration 5: Menu nouveau projet

La boîte de dialogue de création d'un nouveau projet s'affiche. Dans l'onglet **GUI** choisissez **wxWidgets X.X.X Ansi statique application**, saisissez *MyFTP* comme nom de projet puis cliquez sur le bouton **Ok**.

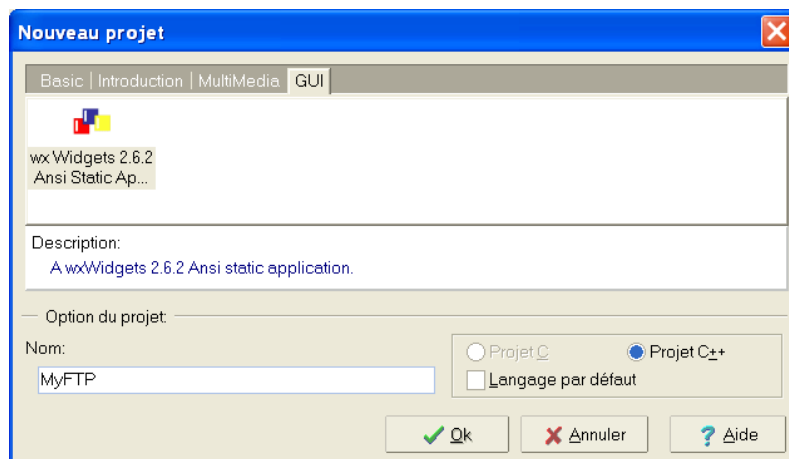


Illustration 6: Dialogue de création d'un nouveau projet

Dans la boîte de dialogue de sauvegarde du projet, créez un nouveau dossier que vous nommerez *Tutoriel wxWidgets* ouvrez ce nouveau dossier puis sauvegardez votre projet en cliquant sur le bouton **Enregistrer**.

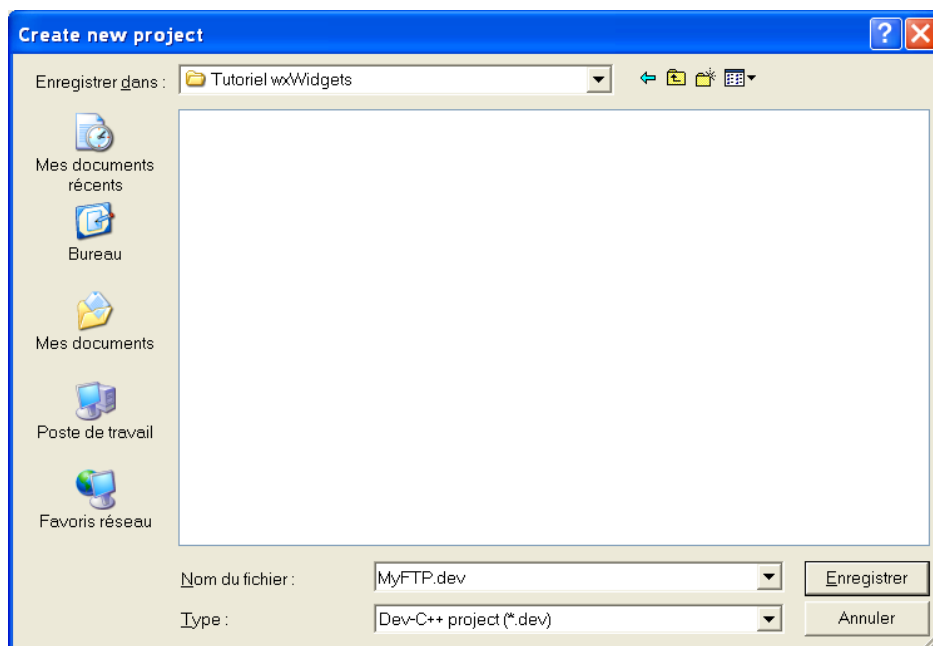


Illustration 7: Dialogue de sauvegarde du projet

Nous n'avons pas besoin du fichier `main.cpp` qui est créé par le template de projet. Nous allons donc le supprimer en choisissant l'item **Supprimer du projet** du menu **Projet**.

Sélectionnez dans la liste `main.cpp` comme indiqué dans l'illustration 8, puis cliquez sur le bouton **Ok**. A la question « Sauvegarder les changements de `main.cpp`? » répondez **No**.

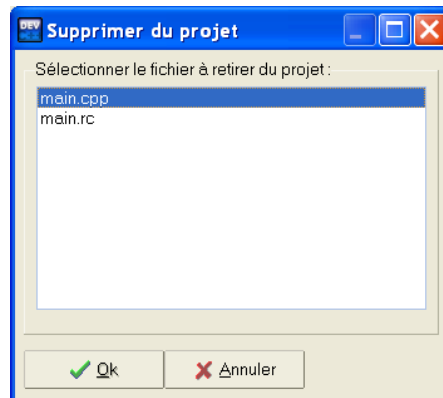


Illustration 8: Dialogue de suppression du projet

Sauvegardez votre projet en choisissant dans le menu **Fichier**, l'item **Sauvegarder**. Vous pouvez aussi utiliser le raccourci clavier **[CTRL+S]**. Dans le dialogue de sauvegarde vérifiez bien que le répertoire actif est *Tutoriel wxWidgets* puis cliquez sur **Enregistrer**.

La classe application

Nous allons créer à présent une classe application dérivée de *wxApp*, nommée *wxMyFTPApp*.

Sélectionnez **Fichier source** dans le sous menu **Nouveau** du menu **Fichier**, ou utilisez le raccourci clavier **[CTRL+N]**. Répondez Yes au message « Ajouter nouveau fichier au projet? » qui s'affiche. Puis faites **[CTRL+S]** pour sauvegarder en donnant au fichier le nom *myftpapp.cpp*.

Dans chaque application wxWidgets vous devrez créer une nouvelle classe application dérivée de *wxApp*. Au minimum cette classe devra redéfinir la fonction membre *OnInit*, c'est à l'intérieur de cette fonction que la fenêtre principale de l'application sera créée.

Créez un autre fichier comme indiqué précédemment mais répondez No au message d'ajout au projet, puis sauvegardez ce fichier sous le nom *myftpapp.h*.

Dans le fichier *myftpapp.h* tapez les lignes suivantes puis sauvegardez :

```
#ifndef _MYFTPAPP_H_
#define _MYFTPAPP_H_

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

```
DECLARE_APP(wxMyFTPApp)

#endif // _MYFTPAPP_H_
```

Les deux premières et la dernière lignes permettent d'éviter les inclusions multiples. Vous trouverez ce genre de lignes dans la plupart des fichiers *.h* en C/C++. La macro `DECLARE_APP` vous permet de définir une fonction `wxGetApp()` renvoyant une référence à votre classe `wxMyFTPApp`.

Dans le fichier `myftpapp.cpp` tapez les lignes suivantes puis sauvegardez :

```
#include "wx/wxprec.h"

#ifndef WX_PRECOMP
#include "wx/wx.h"
#endif

#include "myftpapp.h"

IMPLEMENT_APP(wxMyFTPApp)

wxMyFTPApp::wxMyFTPApp() : wxApp()
{
}

bool wxMyFTPApp::OnInit()
{
    wxApp::OnInit();

    return true;
}
```

Nous voilà donc avec un constructeur vide et une fonction membre qui ne fait pas grand chose à part appeler la fonction *OnInit* de son parent et retourner *true* pour indiquer que tout c'est bien passé. Ne vous inquiétez pas nous allons bientôt compléter ces fonctions avec notamment la création de la fenêtre principale de l'application.

La classe de fenêtre principale de votre application

Notre classe de fenêtre principale sera dérivée de *wxFrame*, cette nouvelle classe aura pour nom *wxMyFTPFrame*. Créez, comme nous l'avons vu précédemment, deux nouveaux fichiers, l'un nommé *myftpframe.cpp* que vous inclurez au projet et l'autre non inclus au projet dont le nom sera *myftpframe.h*.

Dans le fichier *myftpframe.h* saisissez les lignes suivantes :

```
#ifndef _MYFTPFRAME_H_
#define _MYFTPFRAME_H_

class wxMyFTPFrame: public wxFrame
{
public:

    wxMyFTPFrame(wxWindow* parent, wxWindowID id = -1,
                  const wxString& caption = _("myFTP v0.1"),
                  const wxPoint& pos = wxDefaultPosition,
                  const wxSize& size = wxDefaultSize,
```

```

        long style = wxDEFAULT_FRAME_STYLE);

~wxMyFTPFrame();

bool Create(wxWindow* parent, wxWindowID id = -1,
            const wxString& caption = _("myFTP v0.1"),
            const wxPoint& pos = wxDefaultPosition,
            const wxSize& size = wxDefaultSize,
            long style = wxDEFAULT_FRAME_STYLE);

};
#endif // _MYFTPFRAME_H_

```

Nous avons déclaré un constructeur, un destructeur et une fonction membre *Create*. Nous allons maintenant implémenter les fonctions que nous venons de définir.

Dans le fichier *myftpframe.cpp* saisissez les lignes suivantes :

```

#include "wx/wxprec.h"

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#include "myftpframe.h"

wxMyFTPFrame::wxMyFTPFrame(wxWindow* parent, wxWindowID id, const wxString& caption,
                           const wxPoint& pos, const wxSize& size, long style)
{
    Create(parent, id, caption, pos, size, style);
}

wxMyFTPFrame::~wxMyFTPFrame()
{
}

bool wxMyFTPFrame::Create(wxWindow* parent, wxWindowID id, const wxString& caption,
                          const wxPoint& pos, const wxSize& size, long style)
{
    wxFrame::Create(parent, id, caption, pos, size, style);

    return TRUE;
}

```

Que trouvons-nous de nouveau dans ce fichier :

- Le constructeur de la classe, à l'intérieur duquel est appelé la fonction membre *Create*.
- Le destructeur de la classe, qui est vide pour l'instant.
- La fonction membre *Create* qui appelle la fonction *Create* de son parent.

Nous allons légèrement modifier le fichier *myftpapp.cpp* pour que nous puissions compiler notre application et voir s'afficher la fenêtre principale de notre application.

Nous allons inclure le fichier *myftpframe.h* en ajoutant au fichier

```

#include "myftpframe.h"

```

juste après la dernière ligne `#include`, puis dans la fonction membre `OnInit` entre les deux lignes déjà présentes ajouter les lignes suivantes :

```

wxMyFTPFrame* mainWindow = new wxMyFTPFrame(NULL);
mainWindow->SetSize(wxSize(640, 480));
mainWindow->Show(true);

```

Avec ces trois lignes nous créons une instance de l'objet `wxMyFTPFrame`, nous lui donnons une taille de 640 par 480, puis nous l'affichons.

Pour compiler notre application de base nous choisissons l'option **Compiler** du menu **Exécuter** ou nous appuyons sur **[CTRL+F9]**. Si vous avez bien suivi toutes les instructions la compilation se passe sans problème.

Nous pouvons enfin lancer notre application en sélectionnant dans le menu **Exécuter** l'item de menu **Exécuter** ou en utilisant le raccourci clavier **[CTRL+F10]**. La fenêtre qui s'affiche devrait ressembler à celle de l'illustration 9. Vous vous dites qu'elle n'a pas grand chose à voir avec l'illustration 1 à part le titre. Rassurez vous, vous n'êtes qu'au début de ce tutoriel et vous verrez votre application évoluer tout au long de cet ouvrage.



Illustration 9: Fenêtre principale de l'application



Les fichiers sources de ce chapitre sont contenus dans l'archive *Chapitre_3.zip*.

4

Le menu principal de votre application

Dans ce chapitre nous allons aborder la création des menus avec wxWidgets. Nous allons créer une fonction membre de la classe *wxMyFTPFrame* qui contiendra les instructions de création du menu. Cette fonction sera nommée *CreateMenu*.

Il faut donc ajouter dans le fichier *myftpframe.h* à l'intérieur de la classe *wxMyFTPFrame* la déclaration de cette fonction. Pour cela nous créons une section *protected* à la classe et nous ajoutons dans cette section la déclaration comme ci dessous.

```
protected:  
  
void CreateMenu();
```

À la fin du fichier *myftpframe.cpp* nous ajoutons l'implémentation de la fonction *CreateMenu* comme ceci.

```
void wxMyFTPFrame::CreateMenu()  
{  
  
}
```

La barre de menu de l'application est créée avec la classe *wxMenuBar*, elle se compose de quatre menu :

- File
- Local
- Remote
- Help

Ces menus représentés par des objets *wxMenu*, sont ajoutés à la barre de menu par la fonction membre *Append* de *wxMenuBar*. La définition de cette fonction est la suivante :

```
bool Append(wxMenu *menu, const wxString& title)
```

Elle prend comme paramètres un pointeur sur un objet *wxMenu* et une chaîne *wxWidgets* (*wxString*) représentant le texte visible sur la barre de menu. Cette fonction retourne *true* si la création s'est passée sans problème et *false* dans le cas contraire.

Menu File

Ce menu se compose de trois éléments :

- Connecting
- Disconnecting
- Quit

Nous ajouterons ces éléments aussi avec la fonction *Append* mais cette fois ci de la classe *wxMenu* qui est un peu différente. Si vous regardez dans l'aide de *wxWidgets* et recherchez cette fonction, vous verrez qu'elle est définie de trois manières différentes. Dans notre programme nous utiliserons uniquement la première définition qui est celle ci :

```
wxMenuItem* Append(int id, const wxString& item, const wxString& helpString =  
"", wxItemKind kind = wxITEM_NORMAL).
```

Cette fonction renvoie un pointeur sur l'élément ajouté, elle prend comme paramètres, un entier pour l'identification de l'élément dans la gestion des événements, une chaîne pour afficher le texte de l'élément, une autre chaîne pour afficher une aide minimale pour cet élément de menu et un type *wxItemKind* qui définit le type d'élément. Les deux derniers paramètres sont déjà définis par défaut nous pouvons donc, si ceux par défaut conviennent, ne pas les mentionner.



Le type *wxItemKind* peut prendre quatre valeurs différentes : *wxITEM_SEPARATOR*, *wxITEM_NORMAL*, *wxITEM_CHECK* ou *wxITEM_RADIO*. La première est pour créer un séparateur, mais il est plus facile d'utiliser la fonction *AppendSeparator*. La seconde est la valeur courante pour définir un élément simple. Les deux dernières permettent de créer des éléments à cocher et des éléments à sélection unique.

Maintenant retournons à l'intérieur de la fonction *CreateMenu* que nous avons créée plus haut et ajoutons la ligne ci dessous afin de créer l'objet *wxMenuBar*.

```
wxMenuBar* menuBar = new wxMenuBar;
```

Il nous faut à présent créer un objet *wxMenu* et lui ajouter ses trois éléments comme ceci :

```
m_MnuFile = new wxMenu;  
m_MnuFile->Append(ID_MNU_CONNECTING, _("&Connecting..."));  
m_MnuFile->Append(ID_MNU_DISCONNECTING, _("&Disconnecting"));  
m_MnuFile->AppendSeparator();  
m_MnuFile->Append(ID_MNU_QUIT, _("&Quit"));
```

Enfin nous ajoutons ce menu à la barre de menu par la ligne suivante :

```
menuBar->Append(m_MnuFile, _("&File"));
```

Vous avez sans doute remarqué deux choses. La première est l'utilisation de la

fonction membre *AppendSeparator* de la classe *wxMenu*, celle ci permet de séparer deux éléments de menu par une ligne horizontale. La deuxième est que nous employons des variables qui n'ont pas encore été définies, ne vous inquiétez pas nous allons les définir maintenant dans le fichier *myftpframe.h*. Ajoutez dans la déclaration de la classe *wxMyFTPFrame* une section *private*, puis saisissez les lignes suivantes dans cette section.

```
wxMenu* m_MnuFile;  
wxMenu* m_MnuLocal;  
wxMenu* m_MnuRemote;
```

Dans le fichier *myftpframe.cpp* ajoutez les lignes de définition des identificateurs comme ci-dessous, avant l'implémentation du constructeur de *wxMyFTPFrame*.

```
#define ID_MNU_CONNECTING      10201  
#define ID_MNU_DISCONNECTING  10202  
#define ID_MNU_QUIT           10203  
#define ID_MNU_UPLOAD         10301  
#define ID_MNU_CHANGE_LOCAL   10302  
#define ID_MNU_CREATE_LOCAL    10303  
#define ID_MNU_DEL_LOCAL      10304  
#define ID_MNU_REFRESH_LOCAL  10305  
#define ID_MNU_DOWNLOAD       10401  
#define ID_MNU_CREATE_REMOTE  10402  
#define ID_MNU_DEL_REMOTE     10403  
#define ID_MNU_RENAME_REMOTE  10404  
#define ID_MNU_REFRESH_REMOTE 10405  
#define ID_MNU_ABOUT          10501
```

Menu Local

Le menu Local se compose des cinq éléments suivants :

- Upload files
- Change directory...
- Create &directory...
- Erase files
- Refresh

Ajoutons les lignes suivantes à la fonction *CreateMenu* afin de créer ce menu.

```
m_MnuLocal = new wxMenu;  
m_MnuLocal->Append(ID_MNU_UPLOAD, _("&Upload files"));  
m_MnuLocal->Append(ID_MNU_CHANGE_LOCAL, _("&Change directory..."));  
m_MnuLocal->Append(ID_MNU_CREATE_LOCAL, _("&Create &directory..."));  
m_MnuLocal->Append(ID_MNU_DEL_LOCAL, _("&Erase files"));  
m_MnuLocal->Append(ID_MNU_REFRESH_LOCAL, _("&Refresh"));  
menuBar->Append(m_MnuLocal, _("&Local"));
```

Vous avez sûrement remarqué que certains éléments avaient trois points de suspensions à la fin. C'est une convention dans les interfaces qui permet d'indiquer que cet élément va déboucher sur une saisie de l'utilisateur.

Menu Remote

Cinq éléments aussi pour ce menu :

- Download files

- Create directory...
- Erase files
- Rename file...
- Refresh

Le code source pour ajouter ce menu est le suivant, mais je suis sûr que vous allez d'abord essayer de l'ajouter par vous même... Alors ne regardez pas tout de suite la solution saisissez les lignes de codes puis comparez avec celles ci-dessous.

```
m_MnuRemote = new wxMenu;  
m_MnuRemote->Append(ID_MNU_DOWNLOAD, _("&Download files"));  
m_MnuRemote->Append(ID_MNU_CREATE_REMOTE, _("&Create directory..."));  
m_MnuRemote->Append(ID_MNU_DEL_REMOTE, _("&Erase files"));  
m_MnuRemote->Append(ID_MNU_RENAME_REMOTE, _("&Rename file..."));  
m_MnuRemote->Append(ID_MNU_REFRESH_REMOTE, _("&Refresh"));  
menuBar->Append(m_MnuRemote, _("&Remote"));
```

Menu Help

Le menu Help nous servira uniquement à afficher la boîte de dialogue « A Propos de... », le code suivant nous permet de créer cet élément unique de ce menu.

```
wxMenu* MnuHelp = new wxMenu;  
MnuHelp->Append(ID_MNU_ABOUT, _("&About..."));  
menuBar->Append(MnuHelp, _("&Help"));
```

Finition du menu

Maintenant que nous avons créé notre menu, il faut l'ajouter à la fenêtre principale de notre application. Ceci s'effectue avec la fonction membre *SetMenuBar* de la classe *wxFrame*.

A la fin de notre fonction *CreateMenu* nous ajoutons le ligne suivante :

```
SetMenuBar(menuBar);
```

Avant de créer l'appel de la fonction *CreateMenu*, nous allons ajouter une petite décoration à notre fenêtre principale : Une barre de statut. Ce nouvel élément est créé avec l'objet *wxWidgets wxStatusBar*, cet objet est ensuite ajouté à la fenêtre de l'application par la fonction membre *SetStatusBar* de *wxFrame*.

```
wxStatusBar* StatusBar = new wxStatusBar(this, ID_STATUSBAR, wxST_SIZEGRIP|wxNO_BORDER);  
StatusBar->SetFieldsCount(3);  
SetStatusBar(StatusBar);
```

Enfin nous pouvons ajouter à l'intérieur de la fonction *Create* de la classe *wxMyFTPFrame* juste après l'appel de la fonction *Create* de son parent, l'appel de la fonction *CreateMenu*.

```
CreateMenu();
```

Vous pouvez à présent reconstruire votre application, ceci est fait en utilisant la combinaison de touche **[CTRL+F11]** ou par l'élément de menu **Tout Reconstruire** du menu **Exécuter**. La compilation doit se faire sans erreur, puis vous exécutez votre application avec **[CTRL+F10]**. Vous devriez obtenir une fenêtre similaire à l'illustration 10. Dev-Cpp n'étant pas exempt d'erreur il se peut qu'après une reconstruction totale du projet au moment de l'exécution un message s'affiche vous

indiquant que le source n'a pas encore été compilé. Pas de panique ce problème est rapidement réglé en exécutant votre application par la touche [F9].

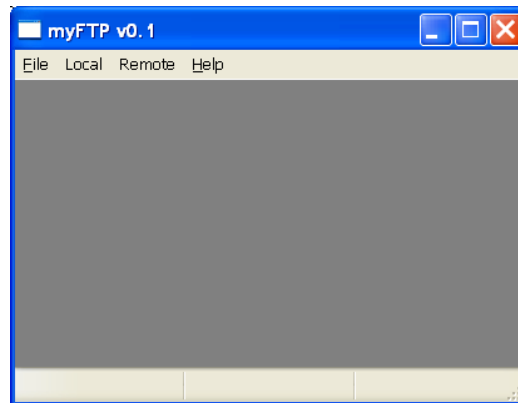


Illustration 10: Fenêtre principale de MyFTP



Les fichiers sources de ce chapitre sont contenus dans l'archive *Chapitre_4.zip*.

5 La barre d'outils

Avant de commencer à étudier ce chapitre, vous devez effectuer quelques manipulations afin d'intégrer à votre projet des images pour votre barre d'outils, et une icône pour votre projet un peu plus jolie que celle fournie par le template. Désarchivez dans le répertoire *Tutoriel wxWidgets* de votre projet le fichier *images.zip*. Les fichiers ci-dessous devraient être créés par cette manipulation.










	myftp_logo.xpm	9 Ko	GIMP image	27/12/2005 15:03	A
	tool_ascii.xpm	3 Ko	GIMP image	27/12/2005 15:50	A
	tool_auto.xpm	2 Ko	GIMP image	27/12/2005 18:13	A
	tool_binary.xpm	2 Ko	GIMP image	27/12/2005 15:49	A
	tool_connecting.xpm	3 Ko	GIMP image	27/12/2005 15:19	A
	tool_disconnecting.xpm	3 Ko	GIMP image	27/12/2005 15:20	A
	tool_download.xpm	2 Ko	GIMP image	27/12/2005 15:25	A
	tool_upload.xpm	2 Ko	GIMP image	27/12/2005 15:24	A
	MyFTP.ico	3 Ko	Icône	23/02/2006 16:22	A

Illustration 11: Liste des fichiers images après désarchivage de images.zip

Dev-Cpp contenant quelques bogues, pour que lors de la prochaine compilation votre nouvelle icône soit prise en compte, vous devrez supprimer les fichiers *MyFTP_private.h*, *MyFTP_private.rc* et *MyFTP_private.res* du répertoire de votre projet.














	main.rc	1 Ko	Resource Sourc...	27/02/2006 17:02	A
	Makefile.win	2 Ko	Fichier WIN	28/02/2006 11:50	A
	MyFTP.dev	2 Ko	Dev-C++ Project...	27/02/2006 18:16	A
	MyFTP.exe	3 052 Ko	Application	28/02/2006 10:53	A
	MyFTP.ico	3 Ko	Icône	23/02/2006 16:22	A
	MyFTP.layout	1 Ko	Fichier LAYOUT	28/02/2006 11:50	A
	myftp_logo.xpm	9 Ko	GIMP image	27/12/2005 15:03	A
	MyFTP_private.h	1 Ko	C Header File	27/02/2006 17:54	A
	MyFTP_private.rc	1 Ko	Resource Sourc...	27/02/2006 17:54	A
	MyFTP_private.res	25 Ko	Fichier RES	28/02/2006 10:49	A
	myftpapp.cpp	3 Ko	C++ Source File	28/02/2006 09:00	A
	myftpapp.h	3 Ko	C Header File	28/02/2006 08:58	A
	myftpapp.o	20 Ko	Fichier O	28/02/2006 10:49	A

Illustration 12: Fichiers à supprimer

Maintenant que tout est prêt, nous allons ajouter une nouvelle fonction membre à la classe *wxMyFTPFrame* qui contiendra les instructions de création de la barre d'outils.

Dans le fichier *myftpframe.h*, après la déclaration de la fonction `CreateMenu` ajoutez la déclaration ci-dessous.

```
void CreateToolBar();
```

Et à la fin du fichier *myftpframe.cpp* ajoutez l'implémentation de la fonction, qui est pour l'instant vide.

```
void wxMyFTPFrame::CreateToolBar()
{
}

```

Puis nous allons de suite ajouter l'appel de cette nouvelle fonction dans la fonction membre `Create` de suite après l'appel de la fonction `CreateMenu`.

```
CreateToolBar();
```



Attention au nom de la fonction et surtout à la casse des lettres. Le `b` de notre fonction `CreateToolBar` est en minuscule, c'est important car une fonction membre de `wxFrame` que nous allons voir dans le chapitre suivant s'appelle `CreateToolBar` avec un `B` majuscule.

La classe `wxToolBar`

La barre d'outils est gérée par une classe `wxToolBar`. Pour créer cet objet nous utiliserons la fonction virtuelle `CreateToolBar` membre de `wxFrame` qui est définie comme indiqué :

```
virtual wxToolBar* CreateToolBar(long style = wxNO_BORDER | wxTB_HORIZONTAL,
wxWindowID id = -1, const wxString& name = "toolBar")
```

Nous assignerons cet instance d'objet `wxToolBar` à notre fenêtre principale par la fonction `SetToolBar` membre de `wxFrame` : `void SetToolBar(wxToolBar* toolBar)`

À tout moment nous pourrons récupérer un pointeur sur l'instance de l'objet `wxToolBar` associé à notre fenêtre par la fonction membre `GetToolBar` :

```
wxToolBar* GetToolBar() const.
```

Pour créer les boutons de notre barre d'outils, nous utiliserons la fonction `AddTool` membre de `wxToolBar`, définie comme suis :

```
wxToolBarToolBase* AddTool(int toolId, const wxString& label, const wxBitmap&
bitmap1, const wxString& shortHelpString = "", wxItemKind kind = wxITEM_NORMAL).
```

Cette fonction renvoie un pointeur sur l'instance de l'objet outil créé et prend en paramètres, l'identificateur de la commande, le texte du bouton, l'image du bouton, un court texte d'aide et le type d'outil. Cela ressemble beaucoup à la fonction de création d'un élément de menu, et c'est assez logique car les événements sont, pour un élément de menu ou un bouton d'un barre d'outil, gérés de manière identique. Mais nous verrons ceci dans les chapitres consacrés aux événements.

Nous pouvons aussi ajouter des séparateurs entre nos outils avec la fonction `AddSeparator` :

```
void AddSeparator().
```

Nous allons enfin commencer à remplir notre fonction `CreateToolbar`, pour débiter nous créons une instance de l'objet `wxToolBar`.

```
wxToolBar* Toolbar = CreateToolBar( wxTB_FLAT|wxTB_HORIZONTAL, ID_TOOLBAR );
```

Puis nous ajoutons notre premier bouton à notre barre d'outils, en saisissant deux nouvelles lignes. La première crée une instance d'objet `wxBitmap` que nous utilisons dans la seconde ligne.

```
wxBitmap BmpConnecting(tool_connecting_xpm);
Toolbar->AddTool(ID_MNU_CONNECTING, wxEmptyString, BmpConnecting, _("Connecting..."), wxITEM_NORMAL);
```

Vous aurez sans doute remarqué le paramètre, `tool_connecting_xpm`, fourni pour la création de l'instance de l'objet `wxBitmap`. Ce paramètre est une variable de type tableau de taille indéfinie sur un pointeur de caractère (`static char * tool_connecting_xpm[]`). Vous vous demandez sans doute où se trouve cette variable dans votre code? En fait elle n'existe pas encore et même quand elle existera vous ne la verrez pas! Pourquoi? Simplement parce qu'elle est définie dans le fichier `tool_connecting.xpm`. Nous allons donc de suite inclure tous les fichiers XPM dans notre fichier `myftpframe.cpp`. Ajoutons les lignes ci-dessous après le dernier `#include`.

```
#include "myftp_logo.xpm"
#include "tool_ascii.xpm"
#include "tool_auto.xpm"
#include "tool_binary.xpm"
#include "tool_connecting.xpm"
#include "tool_disconnecting.xpm"
#include "tool_download.xpm"
#include "tool_upload.xpm"
```



Le format XPM est un format texte que l'on peut afficher avec un éditeur de texte. Vous pouvez essayer d'ouvrir `tool_connecting.xpm` avec le notepad, vous verrez un source en C définissant un tableau de pointeur sur caractères.

Nous en profiterons aussi pour ajouter l'identificateur de notre barre d'outil juste avant celui de la barre de statut.

```
#define ID_TOOLBAR 10101
```

Puis nous ajouterons trois identificateurs pour les radio boutons de la barre d'outils.

```
#define ID_MNU_BINARY 10601
#define ID_MNU_ASCII 10602
#define ID_MNU_AUTO 10603
```

Ajoutons maintenant tous les autres boutons de la barre d'outils, en saisissant le code suivant à la suite de la création du bouton `ID_MNU_CONNECTING` dans le corps de la fonction `CreateToolbar`.

```
    wxBitmap BmpDisconnecting(tool_disconnecting_xpm);
    ToolBar->AddTool(ID_MNU_DISCONNECTING, wxEmptyString, BmpDisconnecting, _("Disconnecting"),
, wxITEM_NORMAL);
    ToolBar->AddSeparator();
    wxBitmap BmpUpload(tool_upload_xpm);
    ToolBar->AddTool(ID_MNU_UPLOAD, wxEmptyString, BmpUpload, _("Upload selected files"),
wxITEM_NORMAL);
    wxBitmap BmpDownload(tool_download_xpm);
    ToolBar->AddTool(ID_MNU_DOWNLOAD, wxEmptyString, BmpDownload, _("Download selected files"),
, wxITEM_NORMAL);
    ToolBar->AddSeparator();
    wxBitmap BmpBinary(tool_binary_xpm);
    ToolBar->AddTool(ID_MNU_BINARY, wxEmptyString, BmpBinary, _("Binary transfert"),
wxITEM_RADIO);
    wxBitmap BmpAscii(tool_ascii_xpm);
    ToolBar->AddTool(ID_MNU_ASCII, wxEmptyString, BmpAscii, _("Ascii transfert"),
wxITEM_RADIO);
    wxBitmap BmpAuto(tool_auto_xpm);
    ToolBar->AddTool(ID_MNU_AUTO, wxEmptyString, BmpAuto, _("Auto transfert"), wxITEM_RADIO);
    ToolBar->AddSeparator();
```

Ensuite, une fois tous nos boutons créés nous appelons le fonction `Realize` membre de `wxToolBar`, puis nous associons la barre d'outils à la fenêtre principale. La dernière ligne sélectionne le bouton radio dont l'identifiant est `ID_MNU_AUTO`.

```
ToolBar->Realize();
SetToolBar(ToolBar);

ToolBar->ToggleTool(ID_MNU_AUTO, true);
```

Comme nous avons inclus toutes les images, nous en profitons pour ajouter à notre fenêtre principale une belle icône dans la barre de titre. Pour faire cela nous ajoutons à la fonction membre `Create` de la classe `wxMyFTPFrame`, juste avant l'instruction `return`, la ligne ci-dessous.

```
SetIcon(wxIcon(myftp_logo_xpm));
```

Vous pouvez reconstruire votre application et l'exécuter, vous obtiendrez une fenêtre correspondante à l'illustration 11.

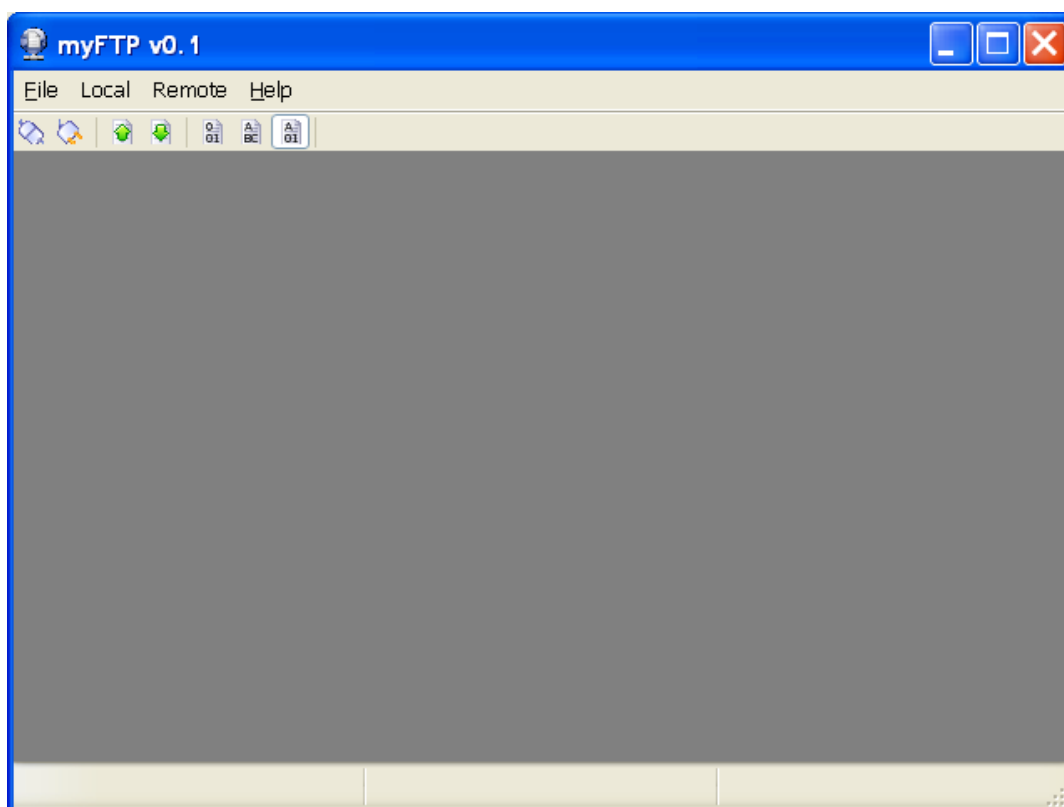


Illustration 13: Fenêtre principale de MyFTP à la fin du chapitre 5



Les fichiers sources de ce chapitre sont contenus dans l'archive *Chapitre_5.zip*.

6

Création des contrôles de l'application

La fenêtre principale de l'application MyFTP est composée de quatre contrôles visibles, un éditeur (*wxTextCtrl*) et trois listes (*wxListCtrl*). Le problème est que ces contrôles doivent suivre les changements de taille de la fenêtre. La bibliothèque wxWidgets met à votre disposition des objets pour résoudre ce problème : les wxSizer.

Les classes wxSizer

Les objets *wxSizer* sont des conteneurs de contrôles qui permettent un placement dynamique sur la fenêtre. L'objet *wxSizer* est une classe de base abstraite, vous ne pouvez pas utiliser directement cette classe, vous devez utiliser une des classes dérivées suivantes :

- wxBoxSizer
- wxStaticBoxSizer
- wxGridSizer
- wxFlexGridSizer
- wxGridBagSizer

Toute la puissance des classes *wxSizer* est dans leur fonction membre Add. Cette fonction se décline de cinq façon différentes.

```
wxSizerItem* Add(wxWindow* window, const wxSizerFlags& flags)
```

```
wxSizerItem* Add(wxWindow* window, int proportion = 0, int flag = 0, int border = 0, wxObject* userData = NULL)
```

```
wxSizerItem* Add(wxSizer* sizer, const wxSizerFlags& flags)
```

```
wxSizerItem* Add(wxSizer* sizer, int proportion = 0, int flag = 0, int border = 0, wxObject* userData = NULL)
```

```
wxSizerItem* Add(int width, int height, int proportion = 0, int flag = 0, int border = 0, wxObject* userData = NULL)
```

Les définitions utilisées le plus souvent sont la deuxième, la quatrième et la cinquième pour ajouter des espace.

Le paramètre *proportion* permet lorsqu'il est égal à 1 d'élargir le contrôle au maximum dans le sens identique au conteneur.

Le paramètre *flag* indique les options du `wxSizer` pour les bordures, l'expansion et l'alignement.

- `WxTOP` : Bordure en haut (valeur indiquée par le paramètre `border`).
- `wxBOTTOM` : Bordure en bas.
- `wxLEFT` : Bordure à gauche.
- `wxRIGHT` : Bordure à droite.
- `wxALL` : Toutes les bordures.
- `wxEXPAND/wxGROW` : Expansion afin d'occuper tout l'espace.
- `wxSHAPED` : Expansion en tentant de garder les proportions.
- `wxALIGN_CENTER` : Centré verticalement et horizontalement.
- `wxALIGN_LEFT` : Alignement à gauche.
- `wxALIGN_RIGHT` : Alignement à droite.
- `wxALIGN_TOP` : Alignement en haut.
- `wxALIGN_BOTTOM` : Alignement en bas.
- `wxALIGN_CENTER_VERTICAL` : Centré verticalement.
- `wxALIGN_CENTER_HORIZONTAL` : Centré horizontalement.

La classe `wxBoxSizer`

C'est la classe la plus simple des `wxSizer`, elle permet de gérer un ensemble de contrôles horizontalement ou verticalement.

La classe `wxStaticBoxSizer`

Cette classe est dérivée de `wxBoxSizer`, la seule différence est qu'elle intègre un objet `wxStaticBox` qui englobera les contrôles contenus dans ce `wxSizer`.

La classe `wxGridSizer`

Un `wxGridSizer` permet de placer les contrôles dans un tableau à deux dimensions (ex : 2 colonnes et 3 lignes). Tous les contrôles bénéficient d'un emplacement de taille identique égale en largeur à la largeur du contrôle le plus large et en hauteur à la valeur du contrôle le plus haut.

La classe `wxFlexGridSizer`

Comme pour le `wxGridSizer` les contrôles sont placés dans un tableau à deux dimensions. La différence se situe dans le fait que chaque colonne peut avoir une largeur différente et chaque ligne peut avoir une hauteur différente.

La classe `wxGridBagSizer`

Cette classe fonctionne un peu comme un `wxFlexGridSizer`, mais permet plus de liberté dans le positionnement avec l'objet `wxGBPosition` et permet un recouvrement avec l'objet `wxGBSpan`. Cette classe utilise des fonctions membres `Add` particulières définies comme ci-dessous.

```
wxSizerItem* Add(wxWindow* window, const wxGBPosition& pos, const wxGBSpan& span = wxDefaultSpan, int flag = 0, int border = 0, wxObject* userData = NULL)
```

```
wxSizerItem* Add(wxSizer* sizer, const wxGBPosition& pos, const wxGBSpan& span = wxDefaultSpan, int flag = 0, int border = 0, wxObject* userData = NULL)
```

```
wxSizerItem* Add(int width, int height, const wxGBPosition& pos, const wxGBSpan& span = wxDefaultSpan, int flag = 0, int border = 0, wxObject* userData = NULL)
```

```
wxSizerItem* Add(wxGBSizerItem* item)
```

Utilisation des `wxSizer` dans l'application

Pour bien comprendre l'utilisation des `wxSizer` dans notre application nous allons étudier le schéma de l'illustration 14. L'emplacement des `wxSizer` est en rouge, celui des contrôles est en bleu à l'exception de l'objet `wxSplitterWindow` qui est en vert.

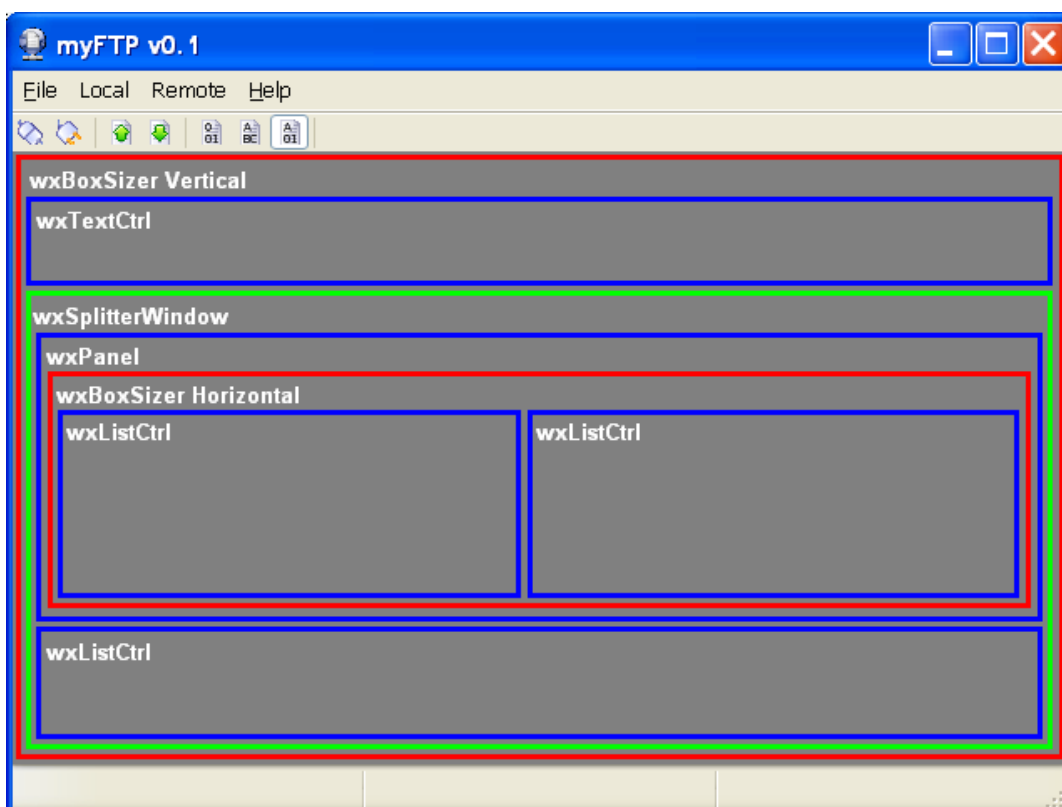


Illustration 14: Schéma de positionnement des contrôles et des wxSizer

Maintenant que nous connaissons le nombre et le type des contrôles que nous allons utiliser, nous devons définir dans le fichier *myftpframe.h* les variables pointeurs qui référenceront ces instances d'objets.

Dans la partie private de la classe MyFTPFrame ajoutons les lignes ci-dessous.

```
wxTextCtrl* m_TextCtrlInfo;  
wxSplitterWindow* m_HorzSplitter;  
wxListCtrl* m_LocalFiles;  
wxListCtrl* m_RemoteFiles;  
wxListCtrl* m_TransfertList;
```

Profitons en pour définir aussi la fonction membre CreateControls en dessous de CreateToolbar comme ceci :

```
void CreateControls();
```

Pour que le compilateur ne génère pas une erreur lorsqu'il rencontre les type *wxListCtrl* et *wxSplitterWindow*, ajoutons avant la définition de la classe *wxMyFTPFrame* les lignes *#include* suivantes :

```
#include <wx/splitter.h>
#include <wx/listctrl.h>
```

A présent à la fin du fichier *myftpframe.cpp* vous devez saisir le squelette de l'implémentation de la fonction *CreateControls* membre *MyFTPFrame* comme vous l'aviez fait pour *CreateToolbar* au chapitre 5 (juste avant le début chapitre 5.1). Ajouter aussi une ligne *#include* pour le fichier *dirctrl.h* (juste avant les lignes *#include* pour les images).

Sous la définition de l'identificateur *ID_STATUSBAR*, saisissez les identificateurs suivants :

```
#define ID_INFORMATION      10103
#define ID_SPLITTERWINDOW   10104
#define ID_FILES_PANEL      10105
#define ID_LOCAL_FILES      10106
#define ID_REMOTE_FILES     10107
#define ID_TRANSFERTS_LIST  10108
```

Nous pouvons enfin compléter l'implémentation de la fonction *CreateControls*. Débutons par la création du *wxBoxSizer* vertical et assignons le à la fenêtre par la fonction *SetSizer* membre de *wxFrame*.

```
wxBoxSizer* Sizer1 = new wxBoxSizer(wxVERTICAL);
SetSizer(Sizer1);
```

Saisissez maintenant le code pour le contrôle *wxTextCtrl* et ajoutons le au *wxBoxSizer* que nous venons de créer avec la fonction *Add* membre de *wxSizer*. Les deuxième et troisième paramètres de la fonction *Add* sont 0 et *wxGROW*. Ceci indique que le contrôle prendra toute la largeur de la fenêtre, même si la taille augmente, mais que sa taille en hauteur n'évoluera pas.

```
m_TextCtrlInfo = new wxTextCtrl(this, ID_INFORMATION, wxEmptyString,
                                wxDefaultPosition, wxSize(-1, 60),
                                wxTE_MULTILINE|wxTE_READONLY);
Sizer1->Add(m_TextCtrlInfo, 0, wxGROW, 0);
```

Faites de même pour le *wxSplitterWindow*, à la différence près que le deuxième paramètre est 1. Le contrôle suivra donc l'augmentation ou la diminution de taille de la fenêtre parente.

```
m_HorzSplitter = new wxSplitterWindow(this, ID_SPLITTERWINDOW,
                                       wxDefaultPosition, wxDefaultSize,
                                       wxSP_3DBORDER|wxSP_3DSASH|wxNO_BORDER);
Sizer1->Add(m_HorzSplitter, 1, wxGROW, 0);
```

Ajoutons les deux contrôles dans le *wxSplitterWindow* et appelons les fonctions nécessaires à leur gestion. La fonction *SetSashGravity* membre de *wxSplitterWindow* permet de définir quel est le ou les contrôles qui devront suivre l'agrandissement ou la diminution de la fenêtre. La fonction *SetMinimumPaneSize* permet de définir un seuil de

taille minimum que les contrôles contenus dans le *wxSplitterWindow* ne pourront pas dépasser. Enfin la fonction *SplitHorizontally* initialise les panneaux haut et bas du contrôle *wxSplitterWindow*.

```
wxPanel* Panell = new wxPanel(m_HorzSplitter, ID_FILES_PANEL, wxDefaultPosition,
                             wxDefaultSize, wxSUNKEN_BORDER|wxTAB_TRAVERSAL);

m_TransfertList = new wxListCtrl(m_HorzSplitter, ID_TRANSFERTS_LIST,
                                wxDefaultPosition, wxDefaultSize, wxLC_REPORT);

m_HorzSplitter->SetSashGravity(1.0);
m_HorzSplitter->SetMinimumPaneSize(60);
m_HorzSplitter->SplitHorizontally(Panell, m_TransfertList);
```

Nous pouvons à présent compléter l'intérieur du *wxPanel* avec le *wxBoxSizer* Horizontal et les deux *wxListCtrl* en saisissant les lignes ci-dessous. Vous remarquerez que les deux contrôles sont ajoutés avec les mêmes paramètres 1 et *wxGrow*. Ils suivront donc les changements de taille du *wxPanel* et se partageront sa largeur à part égale.

```
wxBoxSizer* Sizer2 = new wxBoxSizer(wxHORIZONTAL);
Panell->SetSizer(Sizer2);

m_LocalFiles = new wxListCtrl(Panell, ID_LOCAL_FILES, wxDefaultPosition,
                              wxDefaultSize, wxLC_REPORT|wxRAISED_BORDER|wxLC_VRULES);
Sizer2->Add(m_LocalFiles, 1, wxGROW, 0);

m_RemoteFiles = new wxListCtrl(Panell, ID_REMOTE_FILES, wxDefaultPosition,
                               wxDefaultSize, wxLC_REPORT|wxRAISED_BORDER|wxLC_VRULES);
Sizer2->Add(m_RemoteFiles, 1, wxGROW, 0);
```

Il nous reste à créer les colonnes dans les *wxListCtrl* et assigner une *wxListImage* à celles qui géreront les fichiers et répertoires. Nous utiliserons comme *wxImageList*, celle fourni par un objet *wxFileIconsTable* dont une variable pointeur est déjà définie dans *wWidgets* qui a pour nom *wxTheFileIconsTable*. La fonction membre *GetSmallImageList* de cette classe renvoie un pointeur sur une instance d'objet *wxImageList* qui contient les images correspondantes aux types de fichiers déclarés dans le système.

```
m_TransfertList->InsertColumn(0, _("Job"), wxLIST_FORMAT_LEFT, 80);
m_TransfertList->InsertColumn(1, _("File"), wxLIST_FORMAT_LEFT, 120);
m_TransfertList->InsertColumn(2, _("Address"), wxLIST_FORMAT_LEFT, 120);
m_TransfertList->InsertColumn(3, _("Bytes"), wxLIST_FORMAT_RIGHT, 90);
m_TransfertList->InsertColumn(4, _("Status"), wxLIST_FORMAT_LEFT, 120);
m_TransfertList->InsertColumn(5, _("%"), wxLIST_FORMAT_LEFT, 60);

m_LocalFiles->InsertColumn(0, _("File"), wxLIST_FORMAT_LEFT, 150);
m_LocalFiles->InsertColumn(1, _("Size"), wxLIST_FORMAT_RIGHT, 80);
m_LocalFiles->InsertColumn(2, _("Date"), wxLIST_FORMAT_LEFT, 130);

m_RemoteFiles->InsertColumn(0, _("File"), wxLIST_FORMAT_LEFT, 150);
m_RemoteFiles->InsertColumn(1, _("Size"), wxLIST_FORMAT_RIGHT, 80);
m_RemoteFiles->InsertColumn(2, _("Date"), wxLIST_FORMAT_LEFT, 130);

wxImageList* imageList = wxTheFileIconsTable->GetSmallImageList();
m_LocalFiles->SetImageList(imageList, wxIMAGE_LIST_SMALL);
m_RemoteFiles->SetImageList(imageList, wxIMAGE_LIST_SMALL);
```

L'insertion d'une nouvelle colonne dans un *wxListCtrl* se fait par la fonction

membre `InsertColumn` définie comme ci-dessous.

```
long InsertColumn(long col, const wxString& heading, int format = wxLIST_FORMAT_LEFT, int width = -1)
```

Cette fonction ne peut être utilisée que lorsque la classe `wxListCtrl` est utilisé conjointement avec le paramètre `wxLC_REPORT`. Les paramètres de cette fonction sont les suivants :

- `col` : Index de la colonne
- `heading` : Texte d'entête de colonne
- `format` : Alignement, `wxLIST_FORMAT_LEFT`, `wxLIST_FORMAT_RIGHT` ou `wxLIST_FORMAT_CENTRE`.
- `width` : Largeur de colonne

Il nous reste à ajouter dans la fonction `Create` l'appel de notre fonction `CreateControls` et faire appel aux fonction membres `Fit` et `SetSizeHints` du `wxSizer` assigné à notre fenêtre. Ces deux fonctions permettent la mise en place de tous les contrôles. Puis nous appellerons la fonction `SetSashPosition` membre de `wxSplitterWindow` afin de définir la position du séparateur.

```
CreateControls();  
GetSizer()->Fit(this);  
GetSizer()->SetSizeHints(this);  
m_HorzSplitter->SetSashPosition(65);
```

Après reconstruction et exécution de votre projet vous obtiendrez une fenêtre similaire à celle ci.

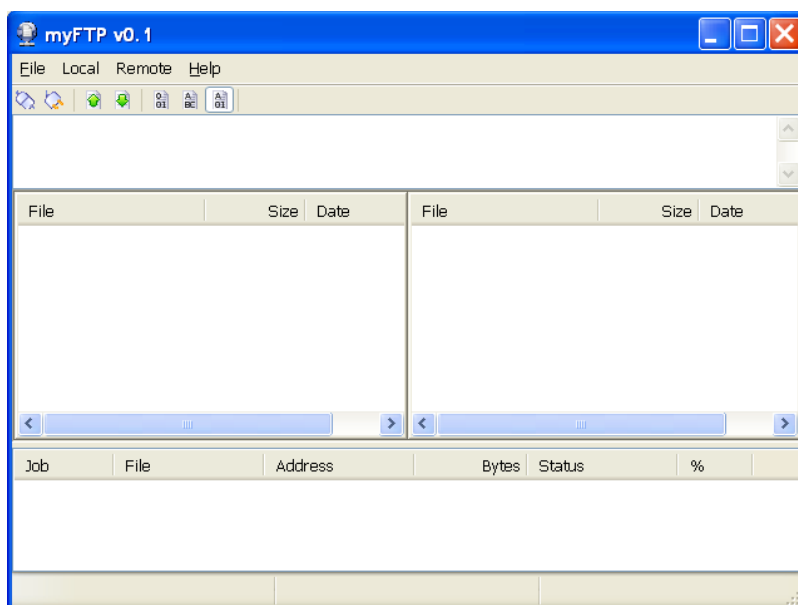


Illustration 15: Fenêtre de MyFTP à la fin du chapitre 6



Les fichiers sources de ce chapitre sont contenus dans l'archive *Chapitre_6.zip*.

7

Une autre classe de fenêtre : wxDialog

Nous avons vu que la fenêtre principale de notre application est dérivée de `wxFrame`. Pour que l'utilisateur puisse saisir les informations de connexion, nous allons créer une boîte de dialogue à l'aide de la classe `wxDialog`.

Différence principale avec entre `wxDialog` et `wxFrame`

Ces deux classe sont issues de `wxWindow`, mais `wxDialog` gère par défaut des événements pour les boutons `wxID_OK`, `wxID_CANCEL` et `wxID_APPLY`.

De plus, la classe `wxDialog` permet à la fenêtre de s'afficher des deux façons :

- Modale avec la fonction membre `int ShowModal()`
- Non modale avec la fonction membre `bool Show(const bool show)`

Création d'une classe dérivée de `wxDialog`

Vous allez maintenant créer un nouveau fichier que vous n'ajouterez pas à votre projet dont le nom sera `cnxdlg.h`. Dans ce fichier nous allons ajouter la déclaration d'une nouvelle classe : `wxMyFTPCnxDlg`. L'utilisateur devra saisir quatre informations dans ce dialogue :

- Le nom du serveur FTP.
- Le numéro du service de connexion (par défaut 21).
- L'identifiant de connexion.
- Le mot de passe associé à l'identifiant.

Ces données seront stockées dans des variables et pourrons être lues et écrites par des accesseurs. Nous allons aussi redéfinir la fonction membre virtuelle `Validate` afin d'ajouter des vérifications lors de la validation.

```
#ifndef _CNXDLG_H_
#define _CNXDLG_H_
```

```
#if defined(__GNUG__) && !defined(__APPLE__)
#pragma interface "cnxdlg.cpp"
#endif

class wxMyFTPCnxDlg: public wxDialog
{
public:
    wxMyFTPCnxDlg(wxWindow* parent, wxWindowID id = -1,
                  const wxString& caption = _("Connecting to"),
                  const wxPoint& pos = wxDefaultPosition,
                  const wxSize& size = wxDefaultSize,
                  long style = wxDEFAULT_DIALOG_STYLE);

    bool Create(wxWindow* parent, wxWindowID id = -1,
               const wxString& caption = _("Connecting to"),
               const wxPoint& pos = wxDefaultPosition,
               const wxSize& size = wxDefaultSize,
               long style = wxDEFAULT_DIALOG_STYLE);

    wxString GetHostname() const {return m_Hostname;}
    void SetHostname(wxString value) {m_Hostname = value;}
    wxString GetService() const {return m_Service;}
    void SetService(wxString value) {m_Service = value;}
    wxString GetUsername() const {return m_Username;}
    void SetUsername(wxString value) {m_Username = value;}
    wxString GetPassword() const {return m_Password;}
    void SetPassword(wxString value) {m_Password = value;}

    virtual bool Validate();

protected:
    void CreateControls();

private:
    wxTextCtrl* m_ctrl_Hostname;
    wxTextCtrl* m_ctrl_Service;

    wxString m_Hostname;
    wxString m_Service;
    wxString m_Username;
    wxString m_Password;
};
#endif // _CNXDLG_H_
```

Créez un nouveau fichier que vous ajouterez à votre projet dont le nom sera *cnxdlg.cpp*, dans ce fichier nous allons ajouter l'implémentation de notre nouvelle classe *wxMyFTPCnxDlg*.

Création des contrôles du dialogue

Voici le schéma de positionnement des contrôles avec les *wxSizer* pour notre dialogue.

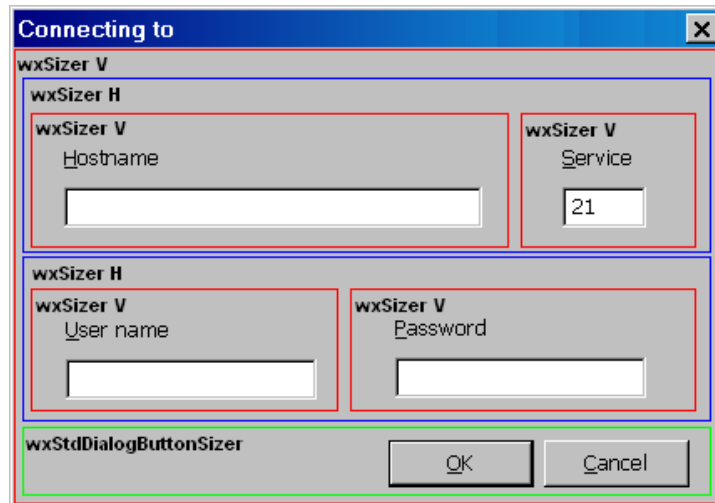


Illustration 16: Positionnement des contrôles avec les wxSizer

L'implémentation de la fonction membre CreateControls correspond à celle ci-dessous:

```
void wxMyFTPCnxDlg::CreateControls()
{
    wxBoxSizer* Sizer1 = new wxBoxSizer(wxVERTICAL);
    SetSizer(Sizer1);

    wxBoxSizer* Sizer2 = new wxBoxSizer(wxHORIZONTAL);
    Sizer1->Add(Sizer2, 0, wxALIGN_LEFT|wxALL, 5);

    wxBoxSizer* Sizer3 = new wxBoxSizer(wxVERTICAL);
    Sizer2->Add(Sizer3, 0, wxALIGN_CENTER_VERTICAL, 0);

    wxStaticText* Text1 = new wxStaticText(this, wxID_STATIC, _("&Hostname"),
                                           wxDefaultPosition, wxDefaultSize, 0);
    Sizer3->Add(Text1, 0, wxALIGN_LEFT|wxALL|wxADJUST_MINSIZE, 5);

    m_ctrl_Hostname = new wxTextCtrl(this, ID_HOSTNAME, wxEmptyString,
                                     wxDefaultPosition, wxSize(250, -1), 0);
    Sizer3->Add(m_ctrl_Hostname, 0, wxALIGN_LEFT|wxALL, 5);

    wxBoxSizer* Sizer4 = new wxBoxSizer(wxVERTICAL);
    Sizer2->Add(Sizer4, 0, wxALIGN_CENTER_VERTICAL, 0);

    wxStaticText* Text2 = new wxStaticText(this, wxID_STATIC, _("&Service"),
                                           wxDefaultPosition, wxDefaultSize, 0);
    Sizer4->Add(Text2, 0, wxALIGN_LEFT|wxALL|wxADJUST_MINSIZE, 5);

    m_ctrl_Service = new wxTextCtrl(this, ID_SERVICE, wxEmptyString,
                                    wxDefaultPosition, wxSize(50, -1), 0);
    Sizer4->Add(m_ctrl_Service, 0, wxALIGN_LEFT|wxALL, 5);

    wxBoxSizer* Sizer5 = new wxBoxSizer(wxHORIZONTAL);
    Sizer1->Add(Sizer5, 0, wxALIGN_LEFT|wxALL, 5);

    wxBoxSizer* Sizer6 = new wxBoxSizer(wxVERTICAL);
    Sizer5->Add(Sizer6, 0, wxALIGN_CENTER_VERTICAL, 0);

    wxStaticText* Text3 = new wxStaticText(this, wxID_STATIC, _("&User name"),
                                           wxDefaultPosition, wxDefaultSize, 0);
    Sizer6->Add(Text3, 0, wxALIGN_LEFT|wxALL|wxADJUST_MINSIZE, 5);

    wxTextCtrl* TextCtrl3 = new wxTextCtrl(this, ID_USER, wxEmptyString,
```

```
        wxDefaultPosition, wxSize(150, -1), 0);
Sizer6->Add(TextCtrl3, 0, wxALIGN_LEFT|wxALL, 5);

wxBoxSizer* Sizer7 = new wxBoxSizer(wxVERTICAL);
Sizer5->Add(Sizer7, 0, wxALIGN_CENTER_VERTICAL, 0);

wxStaticText* Text4 = new wxStaticText(this, wxID_STATIC, _("&Password"),
        wxDefaultPosition, wxDefaultSize, 0);
Sizer7->Add(Text4, 0, wxALIGN_LEFT|wxALL|wxADJUST_MINSIZE, 5);

wxTextCtrl* TextCtrl4 = new wxTextCtrl(this, ID_PASSWORD, wxEmptyString,
        wxDefaultPosition, wxSize(150, -1),
        wxTE_PASSWORD);
Sizer7->Add(TextCtrl4, 0, wxALIGN_LEFT|wxALL, 5);

wxStdDialogButtonSizer* ButtonSizer = new wxStdDialogButtonSizer;

Sizer1->Add(ButtonSizer, 0, wxALIGN_RIGHT|wxALL, 5);
wxButton* BtnOk = new wxButton(this, wxID_OK, _("&OK"), wxDefaultPosition,
        wxDefaultSize, 0);
BtnOk->SetDefault();
ButtonSizer->AddButton(BtnOk);

wxButton* BtnCancel = new wxButton(this, wxID_CANCEL, _("&Cancel"),
        wxDefaultPosition, wxDefaultSize, 0);
ButtonSizer->AddButton(BtnCancel);
ButtonSizer->Realize();
}
```

Bien entendu vous devrez ajouter à votre fichier les déclarations des identificateurs.

```
#define ID_HOSTNAME 10401
#define ID_SERVICE 10402
#define ID_USER 10403
#define ID_PASSWORD 10404
```

Le constructeur de la classe *wxMyFTPCnxDlg* se contentera d'appeler la fonction membre *Create* avec ses paramètres. La fonction *Create* est assez similaire à celle que nous avons écrite pour notre fenêtre principale.

```
bool wxMyFTPCnxDlg::Create(wxWindow* parent, wxWindowID id,
        const wxString& caption, const wxPoint& pos,
        const wxSize& size, long style)
{
    m_ctrl_Hostname = NULL;
    m_ctrl_Service = NULL;

    SetExtraStyle(GetExtraStyle() | wxWS_EX_BLOCK_EVENTS);
    wxDialog::Create(parent, id, caption, pos, size, style);

    CreateControls();
    GetSizer()->Fit(this);
    GetSizer()->SetSizeHints(this);
    Centre();

    return TRUE;
}
```

Contrôle et validation des données

Pour que l'utilisateur ne puisse pas saisir des données autre que celle attendu, il

faut contrôler et valider les données. Ces opérations peuvent se faire de deux manières qui peuvent être utilisées séparément ou conjointement :

- Utilisation de classes `wxValidator`.
- Utilisation de la fonction membre virtuelle `Validate`.

Les classes `wxValidator`

Ces classes sont au nombre de deux : `wxTextValidator` pour tous les contrôles de type `wxTextCtrl`, et `wxGenericValidator` pour les autres contrôles.

Le constructeur de l'objet `wxGenericValidator` ne prend qu'un seul paramètre, un pointeur sur un des types suivants en fonction du contrôle associé :

- `bool` pour un objet `wxCheckBox` ou `wxRadioButton`.
- `wxString` pour un objet `wxButton`, `wxComboBox`, `wxStaticText` ou `wxTextCtrl`.
- `int` pour un objet `wxGauge`, `wxScrollBar`, `wxRadioBox`, `wxSpinButton` ou `wxChoice`.
- `wxArrayInt`. Pour un objet `wxListBox` ou `wxCheckListBox`.

Le constructeur de l'objet `wxTextValidator` prend deux paramètres, le premier est un long qui détermine le style du validateur et un pointeur sur un objet `wxString`.

Neuf styles sont disponibles pour un `wxTextValidator` :

- `wxFILTER_NONE` : Aucun filtre. Tous les caractères sont acceptés.
- `wxFILTER_ASCII` : Les caractères non ASCII sont refusés.
- `wxFILTER_ALPHA` : Les caractères non alpha sont refusés.
- `wxFILTER_ALPHANUMERIC` : Les caractères non alphanumérique sont refusés.
- `wxFILTER_NUMERIC` : Seuls les caractères numériques sont acceptés.
- `wxFILTER_INCLUDE_LIST` : Seul les textes contenus dans la liste d'inclusion sont acceptés.
- `wxFILTER_EXCLUDE_LIST` : Les textes contenus dans la liste d'exclusion sont refusés.
- `wxFILTER_INCLUDE_CHAR_LIST` : Les caractères contenus dans la liste d'inclusion sont acceptés.
- `wxFILTER_EXCLUDE_CHAR_LIST` : Les caractères contenus dans la liste d'exclusion sont refusés.

Les classes de validation sont assignées aux contrôles par la fonction `SetValidator` membre de `wxWindow`. Dans notre application nous utiliserons uniquement la classe

wxTextValidator, avec comme paramètre de style `wxFILTER_NONE` sauf pour le contrôle de saisie du service qui ne devra contenir que des chiffres, donc `wxFILTER_NUMERIC`.

Ajoutez ces lignes à la fin de la fonction *CreateControls* de votre classe *wxMyFTPCnxDlg*.

```
m_ctrl_Hostname->SetValidator(wxTextValidator(wxFILTER_NONE, &m_Hostname));
m_ctrl_Service->SetValidator(wxTextValidator(wxFILTER_NUMERIC, &m_Service));
TextCtrl3->SetValidator(wxTextValidator(wxFILTER_NONE, &m_Username));
TextCtrl4->SetValidator(wxTextValidator(wxFILTER_NONE, &m_Password));
```

La fonction *Validate*

La fonction virtuelle *Validate* est appelée lors de la validation du dialogue. Si le dialogue a le style `wxWS_EX_VALIDATE_RECURSIVELY` positionné, cette fonction appellera la fonction *Validate* de chaque contrôles. Nous implémenterons cette fonction afin qu'elle vérifie si les champs pour le nom du serveur et pour le numéro du service sont bien saisis. Si ce n'est pas le cas la fonction affichera un message d'erreur et retournera *false*.

```
bool wxMyFTPCnxDlg::Validate()
{
    bool ret = wxDialog::Validate();

    if (ret)
    {
        wxString tmp;

        tmp = m_ctrl_Hostname->GetValue();
        if (tmp.IsEmpty())
        {
            wxMessageBox(_("The value of the hostname is compulsory."),
                        _("Error"), wxICON_ERROR|wxOK, this);
            m_ctrl_Hostname->SetFocus();
            ret = false;
        }
        if (ret)
        {
            tmp = m_ctrl_Service->GetValue();
            if (tmp.IsEmpty())
            {
                wxMessageBox(_("The value of the service is compulsory."),
                            _("Error"), wxICON_ERROR|wxOK, this);
                m_ctrl_Service->SetFocus();
                ret = false;
            }
        }
    }

    return ret;
}
```

Vous ne pouvez pas encore voir à quoi ressemble cette boîte de dialogue, mais elle ressemblera à l'illustration ci-dessous.

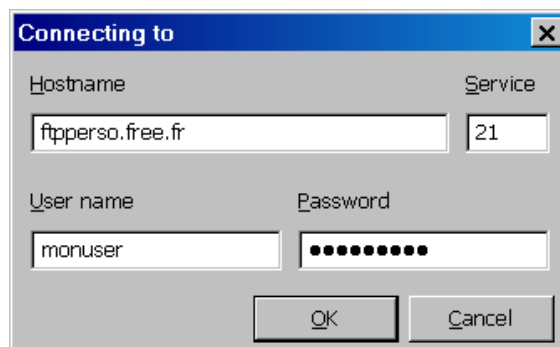


Illustration 17: Le dialogue de connexion



Les fichiers sources de ce chapitre sont contenus dans l'archive *Chapitre_7.zip*.

8

La gestion des événements

Qu'est ce qu'un événement?

Un événement est une action qui intervient dans l'application. Presque tout ce qui survient dans une application est un événement. Le fait que l'utilisateur clique sur un bouton est un événement, la sélection d'un élément de menu est aussi un événement, La fin de période d'un « Timer » est encore un événement.

La bibliothèque wxWidgets nous permet d'associer à un événement particulier survenant dans une classe, un fonction membre de cette classe. Cette fonction ne peut être virtuelle, mais elle sont très similaires. Cette fonction ne prend qu'un seul paramètre, une référence sur une instance d'objet dérivé de *wxEvent*. Cette fonction a un retour de type *void*.

La table des événements

Pour qu'un objet wxWidgets puisse intercepter les événements il faut qu'il soit un descendant de *wxEvtHandler*. La classe *wxWindow* hérite de la classe *wxEvtHandler* donc tous les descendants de *wxWindow* comme *wxPanel*, *wxListCtrl* ou *wxFrame* peuvent intercepter les événements.

Pour définir quels événements doivent être interceptés il faut définir et construire une table des événements. Pour cela nous utilisons la macro *DECLARE_EVENT_TABLE* pour déclarer cette table, et les macros *BEGIN_EVENT_TABLE* et *END_EVENT_TABLE* pour l'implémenter.

Nous allons donc définir notre table des événements pour la classe *wxMyFTPFrame* dans le fichier *myftpframe.h* comme le montre le code ci-dessous en ajoutant *DECLARE_EVENT_TABLE* avant la section *public*.

```
class wxMyFTPFrame: public wxFrame
{
    DECLARE_EVENT_TABLE()

    public:
```

L'implémentation dans le fichier *myftpframe.cpp* est réalisée à l'aide des deux macro citées plus haut. La première macro prend deux paramètres. Le premier est la

classe à laquelle appartient la table, le second est la classe parente. Voici le code pour l'implémentation de la table des événements pour notre classe *wxMyFTPFrame*.

```
BEGIN_EVENT_TABLE(wxMyFTPFrame, wxFrame)

END_EVENT_TABLE()
```

Les événements menu

Pour intercepter un événement menu ou un appuis sur un bouton sur la barre d'outils nous utiliserons des fonctions du type suivant :

```
void (wxEvtHandler::*wxCommandEventFunction)(wxCommandEvent&);
```

Nous allons définir une fonction pour chaque élément de menu, nous ajouterons aussi une fonction pour intercepter la sélection des trois radio-boutons de la barre d'outils, soit quinze fonctions. Nous ajouterons ces fonctions dans la partie *protected* de la définition de la classe *wxMyFTPFrame* dans le fichier *myftpframe.h*.

```
void OnMnuConnectingClick(wxCommandEvent& event);
void OnMnuDisconnectingClick(wxCommandEvent& event);
void OnMnuQuitClick(wxCommandEvent& event);
void OnMnuUploadClick(wxCommandEvent& event);
void OnMnuChangeLocalClick(wxCommandEvent& event);
void OnMnuCreateLocalClick(wxCommandEvent& event);
void OnMnuDelLocalClick(wxCommandEvent& event);
void OnMnuRefreshLocalClick(wxCommandEvent& event);
void OnMnuDownloadClick(wxCommandEvent& event);
void OnMnuCreateRemoteClick(wxCommandEvent& event);
void OnMnuDelRemoteClick(wxCommandEvent& event);
void OnMnuRenameClick(wxCommandEvent& event);
void OnMnuRefreshRemoteClick(wxCommandEvent& event);
void OnMnuAboutClick(wxCommandEvent& event);
void OnMnuTransfertModeClick(wxCommandEvent& event);
```

Maintenant il faut créer le squelette de l'implémentation de ces fonctions dans le fichier *myftpframe.cpp*. Pour chaque fonction cela doit ressembler à ceci :

```
void wxMyFTPFrame::OnMnuConnectingClick(wxCommandEvent& event)
{
    event.Skip();
}
```

Puis nous allons remplir la table des événements, pour cela nous utiliserons la macro *EVT_MENU* qui prend en argument l'identifiant de l'élément de menu et la fonction de gestion de l'événement. La table des événements devrait à présent ressembler à celle ci-dessous. Vous remarquerez la particularité sur la fin de la table, les trois événements *ID_MNU_BINARY*, *ID_MNU_ASCII* et *ID_MNU_AUTO* sont assignés à la même fonction.

```
BEGIN_EVENT_TABLE(wxMyFTPFrame, wxFrame)
    EVT_MENU(ID_MNU_CONNECTING, wxMyFTPFrame::OnMnuConnectingClick)
    EVT_MENU(ID_MNU_DISCONNECTING, wxMyFTPFrame::OnMnuDisconnectingClick)
    EVT_MENU(ID_MNU_QUIT, wxMyFTPFrame::OnMnuQuitClick)
    EVT_MENU(ID_MNU_UPLOAD, wxMyFTPFrame::OnMnuUploadClick)
    EVT_MENU(ID_MNU_CHANGE_LOCAL, wxMyFTPFrame::OnMnuChangeLocalClick)
    EVT_MENU(ID_MNU_CREATE_LOCAL, wxMyFTPFrame::OnMnuCreateLocalClick)
    EVT_MENU(ID_MNU_DEL_LOCAL, wxMyFTPFrame::OnMnuDelLocalClick)
    EVT_MENU(ID_MNU_REFRESH_LOCAL, wxMyFTPFrame::OnMnuRefreshLocalClick)
    EVT_MENU(ID_MNU_DOWNLOAD, wxMyFTPFrame::OnMnuDownloadClick)
    EVT_MENU(ID_MNU_BINARY, wxMyFTPFrame::OnMnuDownloadClick)
    EVT_MENU(ID_MNU_ASCII, wxMyFTPFrame::OnMnuDownloadClick)
    EVT_MENU(ID_MNU_AUTO, wxMyFTPFrame::OnMnuDownloadClick)
```

```

EVT_MENU(ID_MNU_DEL_REMOTE, wxMyFTPFrame::OnMnuDelRemoteClick)
EVT_MENU(ID_MNU_RENAME_REMOTE, wxMyFTPFrame::OnMnuRenameClick)
EVT_MENU(ID_MNU_CREATE_REMOTE, wxMyFTPFrame::OnMnuCreateRemoteClick)
EVT_MENU(ID_MNU_REFRESH_REMOTE, wxMyFTPFrame::OnMnuRefreshRemoteClick)
EVT_MENU(ID_MNU_ABOUT, wxMyFTPFrame::OnMnuAboutClick)
EVT_MENU(ID_MNU_BINARY, wxMyFTPFrame::OnMnuTransfertModeClick)
EVT_MENU(ID_MNU_ASCII, wxMyFTPFrame::OnMnuTransfertModeClick)
EVT_MENU(ID_MNU_AUTO, wxMyFTPFrame::OnMnuTransfertModeClick)

```

```
END_EVENT_TABLE()
```

Pour finir ce chapitre sur les événements menu, nous allons terminer l'implémentation des fonctions `OnMnuQuitClick`, `OnMnuAboutClick`, `OnMnuTransfertModeClick` et `OnMnuConnectingClick`. Pour la première il suffit d'ajouter l'appel à la fonction `Close` membre de la classe `wxWindow`. Et pour la seconde nous utiliserons la fonction d'affichage d'un boîte de message : `wxMessageBox`. Pour la troisième nous ajouterons d'abord, à la définition de la classe `wxMyFTPFrame`, dans la partie private, une variable de type `int` nommée `m_TransfertMode`. Et ajoutons au dessus de la définition de la classe les définitions suivantes :

```

#define TM_ASCII 0
#define TM_BINARY 1
#define TM_AUTO 2

```

Pour la dernière nous devons ajouter l'inclusion du fichier `cnxdlg.h` pour que nous puissions créer une instance de la classe `wxMyFTPCnxDlg`. De plus nous ajouterons dans la partie private de la classe `wxMyFTPFrame` des variables pour stockées les valeurs de connexion. Bien sur ce gestionnaire devra être complété plus tard, pour établir véritablement la connexion au serveur.

```

wxString m_User;
wxString m_Password;
wxString m_HostName;
unsigned short m_Service;

```

```

void wxMyFTPFrame::OnMnuConnectingClick(wxCommandEvent& event)
{
    wxString sService;
    wxMyFTPCnxDlg cnxDlg(this);

    sService = wxString::Format(_T("%hu"), m_Service);
    cnxDlg.SetHostname(m_HostName);
    cnxDlg.SetService(sService);
    cnxDlg.SetUsername(m_User);
    cnxDlg.SetPassword(m_Password);

    if (cnxDlg.ShowModal() == wxID_OK)
    {
        m_HostName = cnxDlg.GetHostname();
        sService = cnxDlg.GetService();
        m_User = cnxDlg.GetUsername();
        m_Password = cnxDlg.GetPassword();

        unsigned long val;
        if (!sService.ToULong(&val) || (val < 1) || (val > 65535))
        {
            wxMessageBox(sService + _T(" is a bad service number."), _("Error"));

```

```
        return;
    }
    m_Service = (unsigned short)val;

}
event.Skip();
}

void wxMyFTPFrame::OnMnuQuitClick(wxCommandEvent& event)
{
    Close();
    event.Skip();
}

void wxMyFTPFrame::OnMnuAboutClick(wxCommandEvent& event)
{
    wxString msg(_("MyFTP Version 0.1.0\n\n"
                  "A simple FTP client program.\n\n"
                  "Copyright (c) Fred Cailleau-Lepetit 2006\n"));

    wxMessageBox(msg, _T("About MyFTP"), wxOK | wxICON_INFORMATION, this);

    event.Skip();
}

void wxMyFTPFrame::OnMnuTransfertModeClick(wxCommandEvent& event)
{
    switch (event.GetInt())
    {
        case ID_MNU_BINARY :
            m_TransfertMode = TM_BINARY;
            break;
        case ID_MNU_ASCII :
            m_TransfertMode = TM_ASCII;
            break;
        case ID_MNU_AUTO :
            m_TransfertMode = TM_AUTO;
            break;
    }
    event.Skip();
}
```

Enfin nous ajouterons au début de la fonction Create membre de la classe wxMyFTPFrame l'affectation de la variable m_TransfertMode.

```
m_TransfertMode = TM_AUTO;
```

A ce stade de l'application vous pouvez lancer une reconstruction puis exécuter votre programme. Testez en choisissant l'élément About du menu Help et vous devriez obtenir un écran ressemblant à celui ci-dessous. Si vous choisissez l'élément Quit du menu File, votre application doit se terminer.

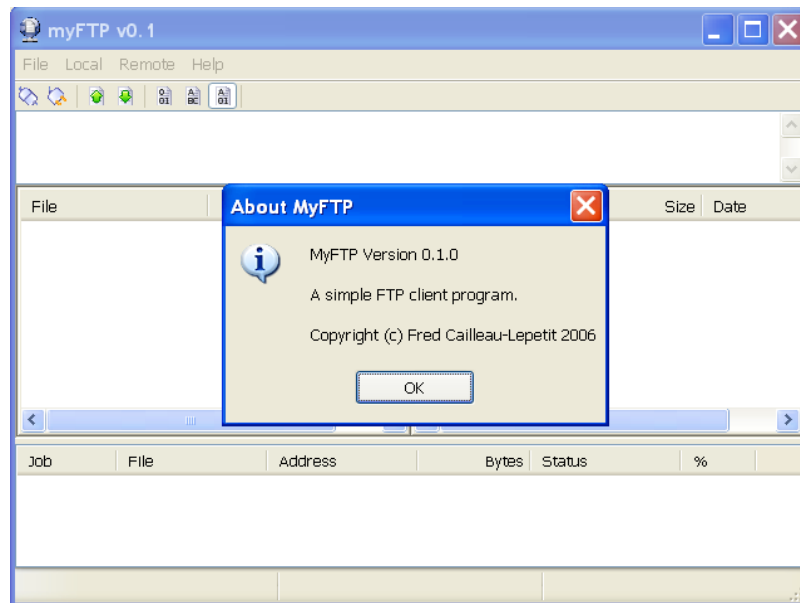


Illustration 18: Exécution de la commande About...

Les événements contrôles

Notre application, contient trois type de contrôles, un `wxTextCtrl`, mais `wxSplitterWindow` et trois `wxListCtrl`.

Les événements de `wxTextCtrl`

Nous n'utiliserons pas les événements spécifiques du contrôle `wxTextCtrl`, mais pour information, ils sont au nombre de 4:

- `wxEVT_COMMAND_TEXT_UPDATED` : Cet événement est généré lorsque le contrôle éditeur change, que ce changement se fasse par l'utilisateur ou par programme (utilisation fonction membre `SetValue`).
- `wxEVT_COMMAND_TEXT_ENTER` : Généré lorsque l'utilisateur appuie sur la touche **[ENTRÉE]**, si le style `wxTE_PROCESS_ENTER` est positionné.
- `wxEVT_COMMAND_TEXT_URL` : Cet événement n'existe que dans les versions `wxMSW` et `wxGTK2`, il est déclenché lorsqu'un événement souris survient sur une URL.
- `wxEVT_COMMAND_TEXT_MAXLEN` : Est déclenché lorsque la limite fixée par la fonction membre `SetMaxlength` est dépassée.

Bien sûr les événements des classes ancêtres peuvent être utilisés, comme par exemple `wxEVT_RIGHT_DOWN` (clic bouton droit de la souris), qui pourrait être utilisé pour affiché un menu contextuel.

Les événements de `wxSplitterWindow`

Ils existe aussi quatre événements spécifiques pour le contrôle `wxSplitterWindow`, mais là non plus nous ne les utiliserons pas dans notre application.

- `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGING` : Est généré pendant la procédure de changement. Peut être utilisé pour mettre à jour l'affichage comme si le traitement prenais fin à cet instant.
- `wxEVT_COMMAND_SPLITTER_SASH_POS_CHANGED` : Cet événement est déclenché lorsque le changement a été effectué. Vous pouvez utiliser cet événement pour par exemple empêcher le changement.
- `wxEVT_COMMAND_SPLITTER_UNSPLOT` : Déclenché lorsque la division a été supprimée.
- `wxEVT_COMMAND_SPLITTER_DOUBLECLICKED` : Généré lorsque le diviseur est « double-cliqué ». Le comportement par défaut supprime la division lorsque c'est possible (absence de dimension minimum pour les panneaux).

Les événements de `wxListCtrl`

Pas moins de vingt événements spécifique pour `wxListCtrl`, nous allons, comme pour les contrôles précédents, les passer tous en revue. Mais nous n'utiliserons dans notre application que quatre de ses événements.

- `wxEVT_COMMAND_LIST_BEGIN_DRAG` : Début d'une opération de Drag&Drop avec le bouton gauche de la souris.
- `wxEVT_COMMAND_LIST_BEGIN_RDRAG` : Début d'une opération de Drag&Drop avec le bouton droit de la souris.
- `wxEVT_COMMAND_LIST_BEGIN_LABEL_EDIT` : Début de l'édition du libellé.
- `wxEVT_COMMAND_LIST_END_LABEL_EDIT` : Fin de l'édition du libellé.
- `wxEVT_COMMAND_LIST_DELETE_ITEM` : Suppression d'un élément.
- `wxEVT_COMMAND_LIST_DELETE_ALL_ITEMS` : Tous les éléments sont supprimés.
- `wxEVT_COMMAND_LIST_ITEM_SELECTED` : Un élément est sélectionné.
- `wxEVT_COMMAND_LIST_ITEM_DESELECTED` : Un élément est désélectionné.
- `wxEVT_COMMAND_LIST_KEY_DOWN` : Une touche a été pressée.
- `wxEVT_COMMAND_LIST_INSERT_ITEM` : Un élément a été inséré.

- `wxEVT_COMMAND_LIST_COL_CLICK` : Une entête de colonne a été « cliquée » avec le bouton gauche de la souris.
- `wxEVT_COMMAND_LIST_ITEM_RIGHT_CLICK` : Un élément a été « cliqué » avec le bouton droit de la souris.
- `wxEVT_COMMAND_LIST_ITEM_MIDDLE_CLICK` : Un élément a été « cliqué » avec le bouton du milieu de la souris.
- `wxEVT_COMMAND_LIST_ITEM_ACTIVATED` : Un élément a été activé par la touche [ENTRÉE] ou par un « double-clic ».
- `wxEVT_COMMAND_LIST_CACHE_HINT` : Préparez le cache pour une liste virtuelle (flags `wxLC_VIRTUAL` et `wxLC_REPORT` positionnés).
- `wxEVT_COMMAND_LIST_COL_RIGHT_CLICK` : Une entête de colonne a été « cliquée » avec le bouton droit de la souris
- `wxEVT_COMMAND_LIST_COL_BEGIN_DRAG` : Début d'une opération de Drag&Drop sur le diviseur de colonne avec le bouton gauche de la souris.
- `wxEVT_COMMAND_LIST_COL_DRAGGING` : Opération de Drag&Drop sur un diviseur de colonne en cours.
- `wxEVT_COMMAND_LIST_COL_END_DRAG` : Fin de l'opération de Drag&Drop sur un diviseur de colonne.
- `wxEVT_COMMAND_LIST_ITEM_FOCUSED` : L'élément actuellement actif a changé.

Nous allons créer les entrées dans la table des événements, les définitions et les squelettes des fonctions pour la gestion des événements suivants :

`wxEVT_COMMAND_LIST_ITEM_SELECTED`,
`wxEVT_COMMAND_LIST_ITEM_DESELECTED`,
`wxEVT_COMMAND_LIST_ITEM_ACTIVATED` et
`wxEVT_COMMAND_LIST_BEGIN_DRAG`. Cela pour les deux `wxListCtrl` définis par les variables `m_LocalFiles` et `m_RemoteFiles`. Nous utiliserons la même fonction membre pour la sélection et la désélection des éléments des deux listes.

Nous ajoutons les entrées ci-dessous dans la table des événements de la classe `wxMyFTPFrame`.

```

EVT_LIST_ITEM_SELECTED(ID_LOCAL_FILES, wxMyFTPFrame::OnFilesChangeSelection)
EVT_LIST_ITEM_DESELECTED(ID_LOCAL_FILES, wxMyFTPFrame::OnFilesChangeSelection)
EVT_LIST_BEGIN_DRAG(ID_LOCAL_FILES, wxMyFTPFrame::OnLocalFilesBeginDrag)
EVT_LIST_ITEM_ACTIVATED(ID_LOCAL_FILES, wxMyFTPFrame::OnLocalFilesItemActivated)
EVT_LIST_ITEM_RIGHT_CLICK(ID_LOCAL_FILES, wxMyFTPFrame::OnLocalFilesRightClick)

EVT_LIST_ITEM_SELECTED(ID_REMOTE_FILES, wxMyFTPFrame::OnFilesChangeSelection)
EVT_LIST_ITEM_DESELECTED(ID_REMOTE_FILES, wxMyFTPFrame::OnFilesChangeSelection)
EVT_LIST_BEGIN_DRAG(ID_REMOTE_FILES, wxMyFTPFrame::OnRemoteFilesBeginDrag)
EVT_LIST_ITEM_ACTIVATED(ID_REMOTE_FILES, wxMyFTPFrame::OnRemoteFilesItemActivated)
EVT_LIST_ITEM_RIGHT_CLICK(ID_REMOTE_FILES, wxMyFTPFrame::OnRemoteFilesRightClick)

```

Maintenant il faut définir les fonctions membres dans le fichier `myftpframe.h`.

```
void OnLocalFilesBeginDrag(wxListEvent& event);
void OnLocalFilesItemActivated(wxListEvent& event);
void OnRemoteFilesBeginDrag(wxListEvent& event);
void OnRemoteFilesItemActivated(wxListEvent& event);
void OnFilesChangeSelection(wxListEvent& event);
```

Et créer les squelettes de ces fonctions dans le fichier `myftpframe.cpp`.

```
void wxMyFTPFrame::OnFilesChangeSelection(wxListEvent& event)
{
    event.Skip();
}

void wxMyFTPFrame::OnLocalFilesBeginDrag(wxListEvent& event)
{
    event.Skip();
}

void wxMyFTPFrame::OnLocalFilesItemActivated(wxListEvent& event)
{
    event.Skip();
}

void wxMyFTPFrame::OnRemoteFilesBeginDrag(wxListEvent& event)
{
    event.Skip();
}

void wxMyFTPFrame::OnRemoteFilesItemActivated(wxListEvent& event)
{
    event.Skip();
}
```

Affichage de la liste locale

Nous faisons un petit aparté pour créer la fonction membre d'affichage des données de la liste locale. Une fois les données affichées nous pourrons continuer à gérer les événements de cette liste. Déclarons dans la partie privée de la classe `wxMyFTPFrame` la fonction *UpdateDisplay* dont le retour sera de type *void* et qui ne prendra aucun paramètre.

Puis dans le fichier `myftpframe.cpp` implémentons cette fonction comme ceci :

```
void wxMyFTPFrame::UpdateDisplay()
{
    int image_id;
    long lCount;
    size_t count;
    wxArrayString files;
    wxString current, tmp;
    DIR *dp;
    dirent *ep;
    struct stat buf;

    current = wxGetCwd();
    m_LocalFiles->DeleteAllItems();

    dp = opendir(current.c_str());
    if (dp != NULL)
    {
        count = 0;
        while ((ep = readdir(dp)) != NULL)
        {
            tmp = ep->d_name;
            files.Add(tmp);
        }
    }
}
```

```

count++; // le Data ne seras pas égal à zéro
if (tmp != _T("."))
{
    if (stat(tmp.c_str(), &buf))
    {
        buf.st_mode = 0;
        buf.st_size = 0;
        buf.st_atime = 0;
        buf.st_mtime = 0;
        buf.st_ctime = 0;
    }
    lCount = m_LocalFiles->GetItemCount();
    lCount = m_LocalFiles->InsertItem(lCount, tmp);

    if (S_ISDIR(buf.st_mode))
    {
        m_LocalFiles->SetItem(lCount, 1, _T("<DIR>"));
        m_LocalFiles->SetItemImage(lCount, wxFileIconsTable::folder);
        m_LocalFiles->SetItemData(lCount, -count);
    }
    else
    {
#ifdef __WXMSW__
        m_LocalFiles->SetItem(lCount, 1, wxString::Format(_T("%lu"), buf.st_size));
    #else
        m_LocalFiles->SetItem(lCount, 1, wxString::Format(_T("%llu"), buf.st_size));
    #endif
        image_id = wxFileIconsTable::file;
        if (tmp.Find(_T('.')) != wxNOT_FOUND)
            image_id = wxTheFileIconsTable->GetIconID(tmp.AfterLast(_T('.')));
        m_LocalFiles->SetItemImage(lCount, image_id);
        m_LocalFiles->SetItemData(lCount, count);
    }
    wxDateTime dateTime(buf.st_mtime);
    // The ISO 8601 format is HH:MM:SS. I don't want the second
    m_LocalFiles->SetItem(lCount, 2, dateTime.FormatISODate() + _T(" ") +
dateTime.FormatISOTime().Left(5));
    }
    closedir (dp);
    m_LocalFiles->SortItems(CompareList, (long)&files);
}
}

```

Vous remarquerez que nous faisons appel à de nouvelles classes wxWidgets. La classe *wxArrayString* est un tableau dynamique d'instance d'objet *wxString*. La fonction *wxGetCwd* renvoie dans un *wxString* le répertoire courant. La classe *wxDateTime* permet la gestion des dates et heures. Notez aussi l'utilisation de fonction de la bibliothèque C, ceci est dû au fait que aucun objet wxWidgets ne permet de gérer facilement la taille et la date des fichiers et répertoires. L'organisation de cette fonction est assez simple. Après avoir « ouvert » le répertoire avec la fonction *opendir*, une boucle est faite tant que la fonction *readdir* ne renvoie pas NULL. A l'intérieur de cette boucle nous ajoutons le nom du fichier au tableau *files*, puis nous récupérons la taille et la date du fichier ou répertoire avec la fonction *stat*. Nous injectons des données dans la liste en fonction du type (fichier ou répertoire), puis utilisons la fonction *SetItemData* membre de *wxListCtrl*, pour indiquer l'index de l'élément et le type en mettant un négatif pour un répertoire. A noter l'utilisation de la fonction *GetIconId* membre de la classe *wxFileIconsTable* pour récupérer l'index de l'image correspondant au type de fichier. En fin de la fonction, « fermeture » du répertoire avec l'appel de la fonction *closedir*, puis tri des éléments avec la fonction membre *SortItems* de *wxListCtrl* en lui

passant comme paramètre l'adresse d'une fonction de comparaison et l'adresse du tableau de wxString.

Pour que notre programme compile correctement nous allons ajouter dans le fichier *myftpframe.cpp* quelques lignes *#include*.

```
#include <sys/stat.h>
#include <dirent.h>
#include <wx/datetime.h>
#include "myftpapp.h"
```

Nous allons déclarer dans le fichier *myftpapp.h* notre fonction de comparaison que nous implémenterons dans le fichier *myftpapp.cpp*.

```
extern int wxCALLBACK CompareList(long item1, long item2, long sortData);
```

```
int wxCALLBACK CompareList(long item1, long item2, long sortData)
{
    wxArrayString* files;

    files = (wxArrayString*)sortData;
    if ((item1 < 0)&&(item2 > 0))
        return -1;
    else if ((item1 > 0)&&(item2 < 0))
        return 1;
    else
        return (*files)[abs(item1) - 1].CmpNoCase((*files)[abs(item2) - 1]);
}
```

Enfin ajoutons l'appel de la fonction *UpdateDisplay* à la fin de la fonction de création des contrôles *CreateControls* membre de *wxMyFTPFrame*. A ce stade si vous compilez et exécutez votre application vous obtiendrez ceci :

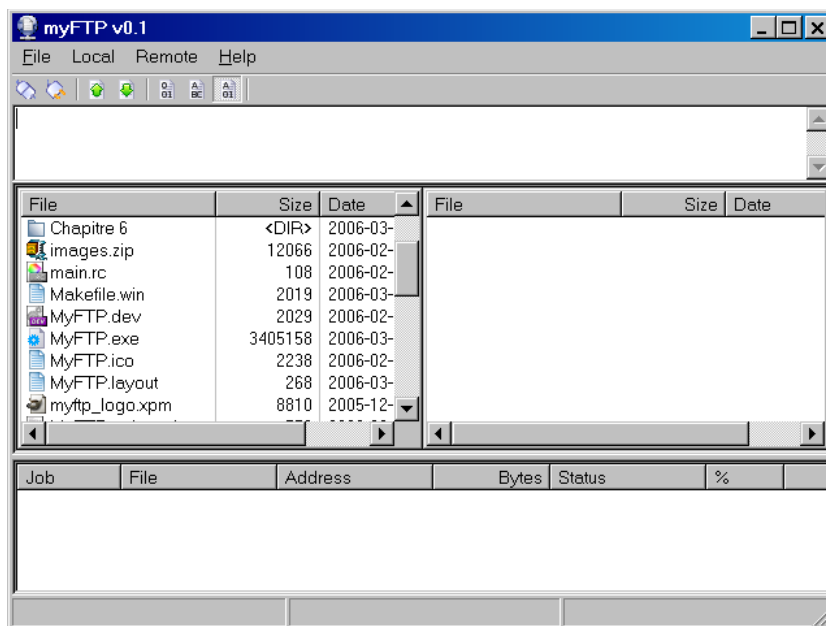


Illustration 19: Affichage des fichiers dans la liste locale

Implémentation de *OnMnuChangeLocalClick*

Cette fonction affichera un boîte de dialogue de sélection de répertoire, grâce à la fonction *wxDirDialog*. Puis au retour valide de celle-ci définira le répertoire de travail avec la fonction *wxSetWorkingDirectory*, Et enfin mettra à jour la liste locale en appelant notre fonction *UpdateDisplay*.

```
void wxMyFTPFrame::OnMnuChangeLocalClick(wxCommandEvent& event)
{
    wxDirDialog DirDlg(this, _("Choose a directory"), wxGetCwd());

    if (DirDlg.ShowModal() == wxID_OK)
    {
        wxSetWorkingDirectory(DirDlg.GetPath());
        UpdateDisplay();
    }

    event.Skip();
}
```

Implémentation de *OnMnuCreateLocalClick*

Pour créer un répertoire local, cette fonction affichera une boîte de dialogue avec la fonction *wxGetTextFromUser*, qui permet de demander une saisie à l'utilisateur. Puis au retour valide de cette fonction, vérification de la saisie de l'utilisateur. Si cette vérification est correcte, création du répertoire avec la fonction *wxMkdir*. Si la création de répertoire se déroule correctement appel de la fonction de mise à jour *UpdateDisplay*. Enfin affichage d'un message d'information dans le *wxTextCtrl* avec sa fonction membre *AppendText*.

```
void wxMyFTPFrame::OnMnuCreateLocalClick(wxCommandEvent& event)
{
    wxString dir, msg;

    dir = wxGetTextFromUser(_("Enter new directory name :"),
                            _("Create a new remote directory"), wxEmptyString, this);
    if (!dir.IsEmpty())
    {
        if (wxMkdir(dir))
        {
            msg = wxString::Format(_("Directory [%s] successfully created."), dir.c_str());
            UpdateDisplay();
        }
        else
        {
            msg = wxString::Format(_("Unable to create directory [%s]."), dir.c_str());
            m_TextCtrlInfo->AppendText(msg);
            m_TextCtrlInfo->AppendText(_T("\n"));
        }
        event.Skip();
    }
}
```

Implémentation de *OnMnuDelLocalClick*

Dans cette fonction nous allons faire deux boucles sur les éléments sélectionnés de la liste avec l'appel de la fonction *GetNextItem* pour récupérer l'index de ces éléments. La première boucle permet de comptabiliser les fichiers et les répertoires en vérifiant la donnée stockée avec l'élément, cette donnée étant négative pour un répertoire. Avec les informations récoltés nous afficherons un message correct pour

demander à l'utilisateur confirmation de la suppression. Enfin avec la deuxième boucle nous effectuerons la suppression des fichiers avec la fonction `wxRemoveFile` et des répertoires avec `wxRmdir`, puis nous afficherons à chaque fois un message sur l'état de la suppression. Enfin nous ferons un appel à la fonction `UpdateDisplay` afin de rafraîchir notre liste.

```
void wxMyFTPFrame::OnMnuDelLocalClick(wxCommandEvent& event)
{
    long item = -1;
    long data;
    wxString name, msg;
    int files, dirs;

    event.Skip();

    files = dirs = 0;
    do
    {
        item = m_LocalFiles->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);

        if (item != -1)
        {
            data = m_LocalFiles->GetItemData(item);
            if (data < 0)
                ++dirs;
            else
                ++files;
        }
    }
    while (item != -1);

    if (files||dirs)
    {
        msg = _("Are you sure you want to delete ");
        if (files)
        {
            if (files > 1)
                msg += _("files ");
            else
                msg += _("file ");
        }
        if (files&&dirs)
            msg += _("and ");
        if (dirs)
        {
            if (dirs > 1)
                msg += _("directories ");
            else
                msg += _("directory ");
        }
        msg += ("?\n");
        if (wxMessageBox(msg, _("Question"), wxYES_NO)==wxNO)
            return;
    }

    do
    {
        item = m_LocalFiles->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);

        if (item != -1)
        {
            name = m_LocalFiles->GetItemText(item);
            data = m_LocalFiles->GetItemData(item);
            if (data < 0)
            {
                if (wxRmdir(name))
```

```

        msg = wxString::Format(_("Directory [%s] successfully deleted."), name.c_str());
    else
        msg = wxString::Format(_("Unable to delete directory [%s]."), name.c_str());
    }
    else
    {
        if (wxRemoveFile(name))
            msg = wxString::Format(_("File [%s] successfully deleted."), name.c_str());
        else
            msg = wxString::Format(_("Unable to delete file [%s]."), name.c_str());
    }
    m_TextCtrlInfo->AppendText(msg);
    m_TextCtrlInfo->AppendText(_T("\n"));
}
while (item != -1);
UpdateDisplay();
}

```

Implémentation de *OnMnuRefreshLocalClick*

Un simple appel de la fonction `UpdateDisplay`.

Implémentation de *OnLocalFilesItemActivated*

Cette fonction permet le changement de répertoire si l'on active un élément qui est un répertoire. Nous allons donc vérifier à l'aide de la fonction membre *GetData* de *wxListCtrl* que la donnée est bien négative. Puis nous changerons de répertoire de travail avec la fonction *wxSetWorkingDirectory* avant de mettre à jour l'affichage avec l'appel de *UpdateDisplay*.

```

void wxMyFTPFrame::OnLocalFilesItemActivated(wxListEvent& event)
{
    wxListItem item = event.GetItem();
    wxString name = item.GetText();
    long l = item.GetData();

    if (l < 0)
    {
        // C'est un répertoire, changer le répertoire courant
        wxSetWorkingDirectory(name);
        UpdateDisplay();
    }
    event.Skip();
}

```

Implémentation de *OnFilesChangeSelection*

Lorsque la sélection est modifiée dans les listes, l'application doit mettre à jour le menu, pour faire cela nous allons créer une fonction spécifique membre de *wxMyFTPFrame* que nous nommerons *EnableMenu*. Nous aurons besoin de savoir si notre client FTP est connecté à ce moment là. Pour cela nous allons ajouter une variable dans la section public de *MyFTPApp* nommée *bConnected* de type booléen. L'implémentation de la fonction *EnableMenu* n'a rien de particulier elle récupère un pointeur sur l'instance de l'objet *wxToolBar* de la fenêtre principale avec la fonction *GetToolBar*, et utilise la fonction membre *GetSelectedItemCount* de *wxListCtrl*, pour connaître le nombre d'élément sélectionnés.

Nous ferons appel à cette nouvelle fonction à trois endroits, d'abord dans la fonction membre *Create* avant l'appel à *SetIcon*, puis dans le gestionnaire d'événement *OnFilesChangeSelection*, et enfin à la fin de la fonction de mise à jour *UpdateDisplay*. Bien sûr, nous n'oublierons pas de définir cette fonction dans la section *public* de la classe *wxMyFTPFrame*, ainsi que d'assigner une valeur à la variable *bConnected* dans le constructeur de *wxMyFTPApp*.

```
void wxMyFTPFrame::EnableMenu()
{
    wxToolBar* toolBar = GetToolBar();
    if (wxGetApp().bConnected)
    {
        int countR = m_RemoteFiles->GetSelectedItemCount();
        int countL = m_LocalFiles->GetSelectedItemCount();
        m_MnuFile->Enable(ID_MNU_CONNECTING, false);
        m_MnuFile->Enable(ID_MNU_DISCONNECTING, true);
        m_MnuLocal->Enable(ID_MNU_UPLOAD, (countL > 0));
        m_MnuRemote->Enable(ID_MNU_DOWNLOAD, (countR > 0));
        m_MnuRemote->Enable(ID_MNU_DEL_REMOTE, true);
        m_MnuRemote->Enable(ID_MNU_RENAME_REMOTE, (countR == 1));
        m_MnuRemote->Enable(ID_MNU_CREATE_REMOTE, true);
        m_MnuRemote->Enable(ID_MNU_REFRESH_REMOTE, true);
        toolBar->EnableTool(ID_MNU_CONNECTING, false);
        toolBar->EnableTool(ID_MNU_DISCONNECTING, true);
        toolBar->EnableTool(ID_MNU_UPLOAD, (countL > 0));
        toolBar->EnableTool(ID_MNU_DOWNLOAD, (countR > 0));
    }
    else
    {
        m_MnuFile->Enable(ID_MNU_CONNECTING, true);
        m_MnuFile->Enable(ID_MNU_DISCONNECTING, false);
        toolBar->EnableTool(ID_MNU_CONNECTING, true);
        toolBar->EnableTool(ID_MNU_DISCONNECTING, false);
        m_MnuLocal->Enable(ID_MNU_UPLOAD, false);
        m_MnuRemote->Enable(ID_MNU_DOWNLOAD, false);
        m_MnuRemote->Enable(ID_MNU_DEL_REMOTE, false);
        m_MnuRemote->Enable(ID_MNU_RENAME_REMOTE, false);
        m_MnuRemote->Enable(ID_MNU_CREATE_REMOTE, false);
        m_MnuRemote->Enable(ID_MNU_REFRESH_REMOTE, false);
        toolBar->EnableTool(ID_MNU_UPLOAD, false);
        toolBar->EnableTool(ID_MNU_DOWNLOAD, false);
    }
}
```

Vous pouvez maintenant reconstruire et exécuter votre application afin de vérifier que le menu est bien mis à jour et que vous pouvez changer de répertoire en « double-cliquant » dessus dans la liste.

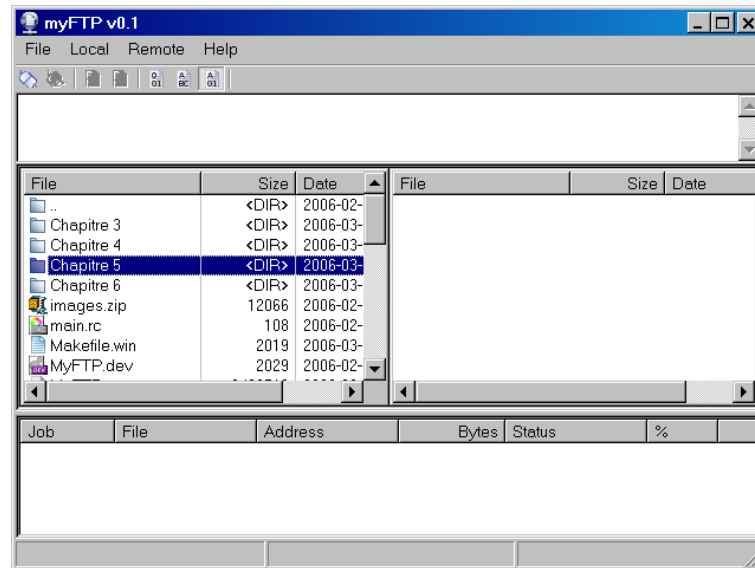


Illustration 20: MyFTP à la fin du chapitre 7



Les fichiers sources de ce chapitre sont contenus dans l'archive *Chapitre_8.zip*.

9

Les sockets avec wxWidgets

Qu'est ce qu'un socket?

Un socket est une sorte de tuyaux dans lequel transite les données sur un réseau. Le contenu du message transporté importe peu au socket, il s'occupe juste d'acheminer le message entre deux éléments supportant les sockets. Pour effectuer ce transport wxWidgets vous fournit la classe `wxSocketBase`. Il est à noter cependant que la classe `wxSocketBase` fonctionne uniquement avec le protocole TCP, le protocole UDP n'étant pas supporté pour l'instant.

Les classes dérivées de wxSocketBase

Le graphique ci-dessous montre les classes issues de la classe `wxSocketBase`. Elles sont au nombre de cinq : `wxSocketServer`, `wxSocketClient`, `wxProtocol`, `wxHTTP` et `wxFTP`.

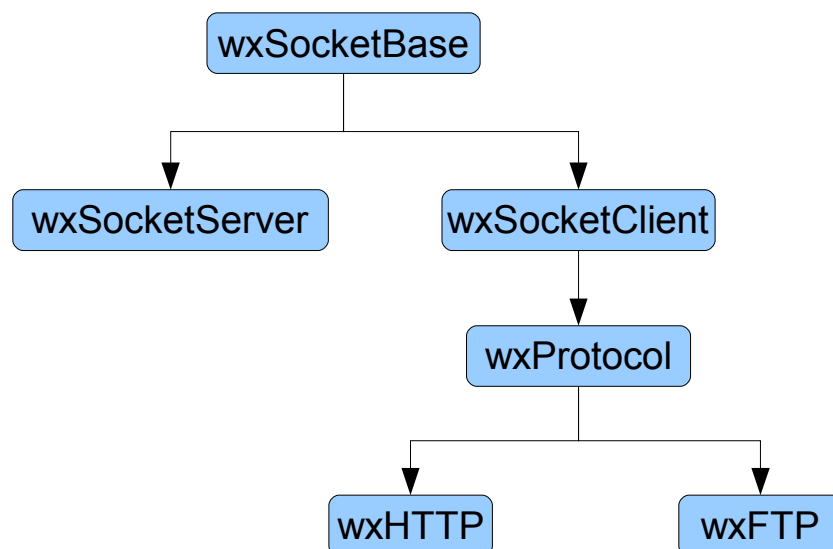


Illustration 21: Hiérarchie des classes dérivées de wxSocketBase

La classe wxSocketServer

Cette classe ajoute des fonctions à la classe wxSocketBase, pour écouter et accepter les connexions sur le port que vous spécifiez dans le constructeur. Voici un exemple très simple de création d'un serveur, d'acceptation d'une connexion et d'envoi d'un message au client connecté.

```
wxIPv4address addr;

addr.Service(3000);

wxSocketServer server(addr);

if (server.Ok()) // Le serveur écoute
{
    if (server.WaitForAccept(30)) // Attendre 30 secondes qu'une connexion arrive
    {
        wxSocketBase* p_socket = server.Accept(); // Accepter la connexion
        wxString msg(_T("HELO\r\n"));
        p_socket->Write(msg.c_str(), msg.Len()); // Envoi d'un message
        if (p_socket->Error()) // Contrôle d'une erreur éventuelle
            wxLogMessage(_T("Erreur lors de l'écriture."));
    }
    server.Close(); // Fermeture du serveur
}
```

Bien sur la construction d'un véritable serveur qui autorise la connexion de multiple clients est bien plus compliquée que cet exemple. Il faudrait gérer l'événement *wxSOCKET_CONNECTION* sur le serveur, les événements *wxSOCKET_INPUT* et *wxSOCKET_LOST* sur les sockets clients qui se connectent et créer un thread pour chaque client connecté.

La Classe wxSocketClient

Les fonctions membres *Connect* et *WaitOnConnect* ont été ajoutées à *wxSocketBase* pour respectivement connecter le client au serveur, dont l'adresse est spécifiée lors de l'appel de *Connect*, et attendre que la connexion soit effectivement établie.

```
wxIPv4address addr;

addr.Hostname(_T("127.0.0.1"));
addr.Service(3000);

wxSocketClient client;

client.Connect(addr, false); // Connexion sur le port 3000 à l'adresse 127.0.0.1
client.WaitOnConnect(10);    // Attente de 10 secondes

if (client.IsConnected())    // Test si la connexion est établie
    wxLogMessage(_T("Connection établie"));
else
{
```

```
client.Close();  
wxLogMessage(_("Connexion impossible"));  
}
```

L'exemple ci-dessus montre comment établir une connexion à un serveur. Pour un exemple un peu plus développé, je vous conseille de regarder dans les exemples fournis avec wxWidgets, vous trouverez dans le sous répertoire sockets des sources un peu plus évoluées.

La classe wxProtocol

La classe wxProtocol n'est pas utilisée directement, elle permet d'avoir une base commune pour wxHTTP et wxFTP. Plusieurs nouvelles fonctions sont introduites dans cette nouvelle classe :

- `bool Reconnect()` : Ferme et réouvre une connexion préalablement établie.
- `wxInputStream* GetInputStream(const wxString& path)` : crée un flux en entrée pour le chemin spécifié.
- `bool Abort()` : Annule la lecture du flux courant.
- `wxProtocolError GetError()` : Retourne le numéro de la dernière erreur survenue.
- `wxString GetContentType()` : Retourne le type mime du dernier flux ouvert.
- `void SetUser(const wxString& user)` : Positionne l'identifiant de l'utilisateur.
- `void SetPassword(const wxString& user)` : Positionne le mot de passe pour l'identifiant courant.
- `wxProtocolError ReadLine(wxString& result)` : Lit une ligne terminée par « \r\n » (CR/LF)

La classe wxHTTP

Cette classe permet de se connecter sur un serveur HTTP. Malheureusement celle-ci est très mal documentée et aucun exemple n'étant fourni avec wxWidgets, je ne pourrai donc pas vous en dire plus que ne le fait le fichier d'aide.

La classe wxFTP

La classe wxFTP permet de se connecter à un serveur FTP et d'effectuer les opérations courantes avec les fonctions membres appropriées.

- void SetPassive(bool pasv) : Active ou désactive le mode passif.
- void SetDefaultTimeout(wxUint32 Value) : Définit la limite de temps par défaut.
- bool SetBinary() : Active le mode de transfert binaire.
- bool SetAscii() : Active le mode de transfert ASCII.
- bool SetTransferMode(TransferMode mode) : Positionne le mode de transfert binaire ou ASCII.
- const wxString& GetLastResult() : Retourne la dernière réponse du serveur.
- char SendCommand(const wxString& command) : Envoie une commande.
- bool CheckCommand(const wxString& command, char expectedReturn) : Envoie une commande et vérifie sa valeur de retour.
- bool ChDir(const wxString& dir) : Change le répertoire courant du serveur.
- bool Mkdir(const wxString& dir) : Crée un répertoire sur le serveur.
- bool Rmdir(const wxString& dir) : Supprime un répertoire sur le serveur.
- wxString Pwd() : Retourne le répertoire courant du serveur.
- bool Rename(const wxString& src, const wxString& dst) : Renomme un fichier ou répertoire sur le serveur.
- bool RmFile(const wxString& path) : Supprime un fichier sur le serveur.
- int GetFileSize(const wxString& fileName) : Retourne la taille d'un fichier.
- bool FileExists(const wxString& fileName) : Vérifie l'existence d'un fichier.
- bool Abort() : Annule un transfert en cours.
- wxInputStream* GetInputStream(const wxString& path) : Retourne un flux en lecture.
- wxOutputStream* GetOutputStream(const wxString& path) : Retourne un flux en écriture.
- bool GetFilesList(wxArrayString& files, const wxString& wildcard) : Retourne la liste des fichiers avec leur chemin, un fichier par ligne.

- `bool GetDirList(wxArrayString& files, const wxString& wildcard)` :
Retourne la liste des fichiers dans un format spécifique au serveur.

Voici un exemple simple de connexion à un serveur FTP et récupération de la liste des fichiers.

```
wxFTP ftp;

if (!ftp.Connect("ftp.wxwindows.org"))
{
    wxLogMessage("Impossible de se connecter!");
}
else
{
    wxArrayString files;
    ftp.ChDir("/pub");
    if (ftp.GetDirList(files))
        for (size_t i = 0; i < files.GetCount(); i++) // Display the files list
            textctrl->AppendText(files.Item(i) + _T("\n"));
}
```

Les événements socket

Les événements des sockets sont au nombre de quatre :

- `wxSOCKET_INPUT` : est déclenché lorsque des données sont prêtes pour être lues.
- `wxSOCKET_OUTPUT` : est déclenché lorsque le socket est prêt pour l'envoi de données. Notamment suite à l'appel de la fonction membre *Connect* ou *Accept* ou à chaque fois que le socket est de nouveau disponible pour l'écriture.
- `wxSOCKET_CONNECTION` : Pour un client, cet événement est déclenché lorsque la connexion est complète. Pour un serveur lorsqu'une demande de connexion arrive de la part d'un client.
- `wxSOCKET_LOST` : Cet événement est reçu lorsque la connexion est cassée ou qu'elle est fermée par l'autre partie. Lorsqu'une demande de connexion échoue cet événement est aussi généré.

Pour la gestion des événements la classe `wxSocketBase` met à notre disposition trois fonction membres :

- `void Notify(bool notify)`
- `void SetNotify(wxSocketEventFlags flags)`
- `void SetEventHandler(wxEvtHandler& handler, int id = -1)`

`Notify` permet suivant la valeur de son paramètre d'activer ou désactiver la génération des événements positionnés par la fonction `SetNotify`. La fonction `SetNotify` permet de définir quels événements seront générés, le paramètre est un élément ou une

combinaison des éléments suivants : `wxSOCKET_INPUT_FLAG`, `wxSOCKET_OUTPUT_FLAG`, `wxSOCKET_CONNECTION_FLAG` et `wxSOCKET_LOST_FLAG`. Enfin `SetEventHandler` permet d'indiquer à quel gestionnaire d'événements sont envoyés les événements générés.

Voici un exemple simple qui démontre l'utilisation des événements avec les sockets.



Attention cependant cet exemple ne démontre pas l'utilisation des sockets, mais comment fonctionne les événements et comment les mettre en place.

Coté client :

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(Minimal_Cnx, MyFrame::OnCnx)
    EVT_SOCKET(SOCKET_ID, MyFrame::OnSocketEvent)
END_EVENT_TABLE()

void MyFrame::OnCnx(wxCommandEvent& event)
{
    // Création du client
    Client = new wxSocketClient();
    // On indique que le gestionnaire d'événement est La fenêtre
    Client->SetEventHandler(*this, SOCKET_ID);
    // Spécification des événements à générer
    Client->SetNotify(wxSOCKET_CONNECTION_FLAG | wxSOCKET_INPUT_FLAG |
                    wxSOCKET_OUTPUT_FLAG | wxSOCKET_LOST_FLAG);
    // les événements peuvent être générés
    Client->Notify(true);

    wxIPv4address addr;
    addr.Hostname("127.0.0.1");
    addr.Service(3000);

    Client->Connect(addr, false);
    Client->WaitOnConnect(10);

    if (Client->IsConnected())
        Info(_("Connexion réussie!"));
    else
    {
        Client->Close();
        Info(_("La connexion à échouée."));
    }
}

void MyFrame::OnSocketEvent(wxSocketEvent& event)
{
    wxSocketBase* ev_sock;
    char buffer[1024];
    wxString str;
    unsigned int count;

    switch(event.GetSocketEvent())
    {
        case wxSOCKET_INPUT :
            Info(_("Des données peuvent être reçues."));
            ev_sock = event.GetSocket();
            ev_sock->SetFlags(wxSOCKET_BLOCK|wxSOCKET_WAITALL);
            // Attention ReadMsg ne peut être utilisé que lorsque le buffer
            // est envoyé avec WriteMsg
            ev_sock->ReadMsg(buffer, sizeof(buffer));
            if (ev_sock->Error())
```

```

        Info(_("Problème lors de la lecture!"));
    else
    {
        count = ev_sock->LastCount();
        Info(wxString::Format(_("%u octets lus"), count));
        str = wxString(buffer, (count < sizeof(buffer) ? count : sizeof(buffer)));
        Info(str);
    }
    break;
case wxSOCKET_OUTPUT :
    Info(_("Des données peuvent être envoyées.));
    ev_sock = event.GetSocket();
    ev_sock->SetFlags(wxSOCKET_BLOCK|wxSOCKET_WAITALL);
    str = _("Test d'écriture");
    // Attention WriteMsg ne peut être utilisé que lorsque le buffer
    // est lu avec ReadMsg
    ev_sock->WriteMsg(str.c_str(), str.Len());
    if (ev_sock->Error())
        Info(_("Problème lors de l'écriture!"));
    else
    {
        count = ev_sock->LastCount();
        Info(wxString::Format(_("%u octets écrits"), count));
    }
    break;
case wxSOCKET_LOST :
    Info(_("La connexion est perdue.));
    break;
case wxSOCKET_CONNECTION :
    Info(_("La connexion est active.));
    break;
default :
    Info(_("Un événement inconnu est arrivé!));
}
}

void MyFrame::Info(const wxString& msg)
{
    CtrlInfo->AppendText(msg);
    CtrlInfo->AppendText(_T("\n"));
}

```

Coté serveur :

```

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(Minimal_Cnx, MyFrame::OnCnx)
    EVT_SOCKET(SERVER_ID, MyFrame::OnServerEvent)
    EVT_SOCKET(SOCKET_ID, MyFrame::OnSocketEvent)
END_EVENT_TABLE()

void MyFrame::OnCnx(wxCommandEvent& event)
{
    wxIPv4address addr;
    addr.Service(3000);

    // Création du serveur
    Server = new wxSocketServer(addr);

    // On teste si le serveur écoute
    if (Server->Ok())
        Info(_("Le serveur écoute!));
    else
        Info(_("le serveur ne peut écouter sur le port spécifié!));

    // On indique que le gestionnaire d'événement est La fenêtre
    Server->SetEventHandler(*this, SERVER_ID);
    // Spécification des événements à générer
    Server->SetNotify(wxSOCKET_CONNECTION_FLAG);
    // les événements peuvent être générés

```

```
Server->Notify(true);
}

void MyFrame::OnServerEvent(wxSocketEvent& event)
{
    wxSocketBase *sock;

    switch(event.GetSocketEvent())
    {
        case wxSOCKET_CONNECTION :
            Info(_("Une demande de connexion est arrivée."));
            sock = Server->Accept(false);
            if (sock)
            {
                Info(_("La connexion est acceptée."));
                // On indique que le gestionnaire d'événement est La fenêtre
                sock->SetEventHandler(*this, SOCKET_ID);
                // Spécification des événements à générer
                sock->SetNotify(wxSOCKET_INPUT_FLAG | wxSOCKET_OUTPUT_FLAG | wxSOCKET_LOST_FLAG);
                // les événements peuvent être générés
                sock->Notify(true);
            }
            else
                Info(_("une erreur est survenue lors de l'acceptation de la connexion."));
            break;
        default :
            Info(_("Un événement inconnu est arrivé!"));
    }
}

void MyFrame::OnSocketEvent(wxSocketEvent& event)
{
    wxSocketBase* ev_sock;
    char buffer[1024];
    wxString str;
    unsigned int count;

    switch(event.GetSocketEvent())
    {
        case wxSOCKET_INPUT :
            Info(_("Des données peuvent être reçues."));
            ev_sock = event.GetSocket();
            ev_sock->SetFlags(wxSOCKET_BLOCK|wxSOCKET_WAITALL);
            // Attention ReadMsg ne peut être utilisé que lorsque le buffer
            // est envoyé avec WriteMsg
            ev_sock->ReadMsg(buffer, sizeof(buffer));
            if (ev_sock->Error())
                Info(_("Problème lors de la lecture!"));
            else
            {
                count = ev_sock->LastCount();
                Info(wxString::Format(_("%u octets lus"), count));
                str = wxString(buffer, (count < sizeof(buffer) ? count : sizeof(buffer)));
                Info(str);
            }
            break;
        case wxSOCKET_OUTPUT :
            Info(_("Des données peuvent être envoyées."));
            ev_sock = event.GetSocket();
            ev_sock->SetFlags(wxSOCKET_BLOCK|wxSOCKET_WAITALL);
            str = _("Un bonjour de la part du serveur!");
            // Attention WriteMsg ne peut être utilisé que lorsque le buffer
            // est lu avec ReadMsg
            ev_sock->WriteMsg(str.c_str(), str.Len());
            if (ev_sock->Error())
                Info(_("Problème lors de l'écriture!"));
            else
            {
                count = ev_sock->LastCount();
                Info(wxString::Format(_("%u octets écrits"), count));
```

```

    }
    break;
case wxSOCKET_LOST :
    Info(_("La connexion est perdue."));
    break;
default :
    Info(_("Un événement inconnu est arrivé!"));
}
}

```

Création d'une classe dérivée de wxFTP

Qu'allons nous modifier à la classe wxFTP pour créer notre nouvelle classe?

- Ajout de la commande CDUP du protocole FTP avec la fonction membre CdUp.
- Gestion, sauvegarde et lecture du *hostname* et du *service* avec un nouveau constructeur, les fonctions Connect, Service et Hostname.
- Lecture du nom de l'utilisateur avec la fonction membre User.
- Lecture de l'état du transfert avec la fonction membre OnStreaming.
- Modification de fonctions pour une gestion séparée du chemin et du nom de fichier. Soit les fonctions GetFileSize, GetOutputStream, GetInputStream et GetLastOpenStreamResult.

Créez un nouveau fichier sans l'ajouter au projet, puis sauvegardez ce fichier sous le nom *myftp.h*. Dans ce fichier nous allons déclarer notre nouvelle classe wxMyFTP comme indiqué ci-dessous.

```

#ifndef _MYFTP_H_
#define _MYFTP_H_

#if defined(__GNUG__) && !defined(__APPLE__)
#pragma interface "myftp.cpp"
#endif
#include <wx/protocol/ftp.h>

class wxMyFTP : public wxFTP
{
public:

    wxMyFTP(const wxString& hostname, unsigned short service) : wxFTP()
    {m_Hostname = hostname; m_Service = service;}

    bool Connect(wxSockAddress& address, bool wait = true);
    bool Connect();

    unsigned short Service(){return m_Service;}
    wxString Hostname(){return m_Hostname;}
    wxString User();

    bool CdUp();
    int GetFileSize(const wxString& path, const wxString& filename);
    wxOutputStream* GetOutputStream(const wxString& path,
                                    const wxString& filename, bool ascii);
    wxInputStream* GetInputStream(const wxString& path,
                                  const wxString& filename, bool ascii);

```

```
const wxString& GetLastOpenStreamResult() {return m_LastOpenStreamResult;}

bool OnStreaming() {return m_streaming;}

private:

    wxString m_LastOpenStreamResult;
    wxString m_Hostname;
    unsigned short m_Service;

    wxMyFTP() : wxFTP() {}

    DECLARE_DYNAMIC_CLASS_NO_COPY(wxMyFTP)
};

#endif // _MYFTP_H
```

Pour implémenter notre nouvelle classe wxMyFTP, nous allons créer un nouveau fichier nommé *myftp.cpp* que nous allons ajouter au projet.

```
#if defined(__GNUG__) && !defined(__APPLE__)
#pragma implementation "myftp.h"
#endif

// For compilers that support precompilation, includes "wx/wx.h".
#include <wx/wxprec.h>

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include <wx/wx.h>
#endif
#include "myftp.h"

IMPLEMENT_DYNAMIC_CLASS(wxMyFTP, wxFTP)

bool wxMyFTP::Connect(wxSocketAddress& address, bool wait)
{
    if (address.IsKindOf(CLASSINFO(wxIPAddress)))
    {
        m_Hostname = ((wxIPAddress*)&address)->Hostname();
        m_Service = ((wxIPAddress*)&address)->Service();
    }
    return wxFTP::Connect(address, wait);
}

bool wxMyFTP::Connect()
{
    wxIPv4address ipAddress;

    ipAddress.Hostname(m_Hostname);
    ipAddress.Service(m_Service);

    return wxFTP::Connect(ipAddress, false);
}

wxString wxMyFTP::User()
{
    if (m_user == _T("anonymous"))
        return wxEmptyString;
    else
        return m_user;
}
```

```

bool wxMyFTP::CdUp()
{
    return wxFTP::DoSimpleCommand(_T("CDUP"));
}

int wxMyFTP::GetFileSize(const wxString& path, const wxString& filename)
{
    int fileSize;

    wxString OldPath = wxFTP::Pwd();    // Sauvegarde du chemin actuel

    wxFTP::ChDir(path);    // Changement du chemin
    fileSize = wxFTP::GetFileSize(filename);
    wxFTP::ChDir(OldPath);    // Restauration du chemin

    return fileSize;
}

wxOutputStream* wxMyFTP::GetOutputStream(const wxString& path,
                                         const wxString& filename, bool ascii)
{
    wxOutputStream* stream;

    wxString OldPath = wxFTP::Pwd();    // Sauvegarde du chemin actuel

    wxFTP::ChDir(path);    // Changement du chemin
    if (ascii)
        SetAscii();        // Mode de transfert Ascii
    else
        SetBinary();        // Mode de transfert Binaire
    stream = wxFTP::GetOutputStream(filename);
    m_LastOpenStreamResult = wxFTP::GetLastResult();
    wxFTP::ChDir(OldPath);    // Restauration du chemin

    return stream;
}

wxInputStream* wxMyFTP::GetInputStream(const wxString& path,
                                       const wxString& filename, bool ascii)
{
    wxInputStream* stream;

    wxString OldPath = wxFTP::Pwd();    // Sauvegarde du chemin actuel

    wxFTP::ChDir(path);    // Changement du chemin
    if (ascii)
        SetAscii();        // Mode de transfert Ascii
    else
        SetBinary();        // Mode de transfert Binaire
    stream = wxFTP::GetInputStream(filename);
    m_LastOpenStreamResult = wxFTP::GetLastResult();
    wxFTP::ChDir(OldPath);    // Restauration du chemin

    return stream;
}

```

L'implémentation de la première fonction `Connect` est assez simple, après vérification que le paramètre *address* est du type *wxIPAddress* les parties *hostname* et *service* sont extraites et sauvegardées, ensuite la fonction `Connect` de la classe parent est appelée.

La fonction `CdUp` appelle la fonction *DoSimpleCommand* de *wxFTP* avec comme paramètre la commande à envoyer c'est à dire `CDUP`.

Les fonctions `GetFileSize`, `GetOutputStream` et `GetInputStream` sont toutes construites sur le même principe : Sauvegarde du chemin actuel avec l'appel de la

fonction *Pwd*, changement du chemin avec la fonction *ChDir* à laquelle on indique le paramètre de chemin fourni, appel de la fonction de base (ou du groupe de fonctions), et enfin restauration du chemin précédemment sauvegardé avec la fonction membre *ChDir*.

Pour terminer ce chapitre nous allons ajouter à notre classe application un tableau de pointeur sur notre classe `wxMyFTP` et deux fonctions pour gérer l'ajout et la récupération des instances d'objet dans ce tableau.

Dans le fichier `myftppapp.h` ajoutez avant la déclaration de la classe application les deux lignes ci-dessous.

```
class wxMyFTP;  
WX_DEFINE_ARRAY_PTR(wxMyFTP*, wxArrayFTP);
```

Dans la déclaration de la classe `wxMyFTPApp` ajoutez dans la partie publique :

```
wxArrayFTP m_ftps;  
  
wxMyFTP* CreateFTP(const wxString& hostname, unsigned short service,  
                  const wxString& user, const wxString& password,  
                  bool passive);
```

et créez une partie `private` comme ci-dessous :

```
private:  
  
wxMyFTP* m_UsedFTP;  
  
wxMyFTP* GetFTP(const wxString& hostname, unsigned short service,  
               const wxString& user);
```

Dans le fichier `myftppapp.cpp` on ajoute une ligne pour inclure le fichier `myftp.h`, puis dans le constructeur de `wxMyFTPApp` on affecte 0 à la variable `m_UsedFTP`. Enfin on implémente les deux fonctions `GetFTP` et `CreateFTP`.

```
wxMyFTP* wxMyFTPApp::GetFTP(const wxString& hostname, unsigned short service, const wxString& user)  
{  
    wxMyFTP* ftp = 0;  
  
    size_t count = m_ftps.GetCount();  
  
    for (size_t i = 0; i < count; i++)  
    { // On boucle dans le tableau tant qu'une instance avec les propriétés  
      // voulues n'est pas trouvée  
      if ((m_ftps[i]->Service() == service)&&  
          (m_ftps[i]->Hostname() == hostname)&&  
          (m_ftps[i]->User() == user))  
      {  
          ftp = m_ftps[i];  
          break;  
      }  
    }  
}
```

```
        return ftp;
    }

wxMyFTP* wxMyFTPApp::CreateFTP(const wxString& hostname, unsigned short service,
                               const wxString& user, const wxString& password, bool passive)
{
    // Une instance avec les bons paramètres existe t-elle?
    wxMyFTP* ftp = GetFTP(hostname, service, user);
    if (!ftp)
    {
        // Non alors on l'alloue et on l'ajoute au tableau
        ftp = new wxMyFTP(hostname, service);
        if (user != wxEmptyString)
        {
            ftp->SetUser(user);
            ftp->SetPassword(password);
        }
        ftp->SetDefaultTimeout(10);
        ftp->SetPassive(passive);
        m_ftps.Add(ftp);
    }
    return ftp;
}
```



Les fichiers sources de ce chapitre sont contenus dans l'archive *Chapitre_9.zip*.

10

Le multi-tâches

Qu'est ce que le multi-tâches?

Une tâche est une partie d'un programme qui s'exécute indépendamment des autres. Les tâches sont parfois appelées des processus légers, mais la différence fondamentale entre des tâches et les processus est que ces derniers utilisent des espaces mémoire séparés alors que les tâches utilisent le même espace mémoire.

Le multi-tâches est donc l'exécution simultanée de plusieurs parties d'un même programme.

Les classes pour la gestion du multi-tâches

Pour gérer le multi-tâches wxWidgets met à notre disposition une collection de huit classes :

- wxThread : classe de tâche.
- wxThreadHelper : classe utilitaire pour la gestion de wxThread.
- wxMutex : classe pour la synchronisation.
- wxMutexLocker : classe utilitaire pour la gestion de wxMutex.
- wxCriticalSection : classe pour la synchronisation.
- wxCriticalSectionLocker : classe utilitaire pour la gestion de wxCriticalSection.
- wxCondition : classe pour la synchronisation.
- wxSemaphore : classe pour la synchronisation.

Pour l'application que nous sommes entrain de développer, nous n'utiliserons que trois d'entre elles qui vont être décrites plus précisément dans les chapitres suivants.

La classe `wxThread`

La classe `wxThread` est une classe de base virtuelle pure. Vous ne pouvez l'utiliser directement, vous devez créer une classe dérivée et implémenter obligatoirement la fonction membre virtuelle pure `Entry`. Cette fonction est le point d'entrée de la tâche lorsqu'elle est exécutée par la fonction `Run`.

```
virtual ExitCode Entry();  
wxThreadError Run();
```

La classe `wxCriticalSection`

Cette classe permet une synchronisation des tâches lors de l'accès à des données partagées. Pour effectuer cette synchronisation `wxCriticalSection` met à notre disposition deux fonctions membre : `Enter` et `Leave`. Les noms de ces fonctions sont assez explicites pour bien comprendre que lorsque que l'on entre dans une portion de code dans laquelle on veut protéger l'accès exclusif à une donnée, on utilise `Enter` puis on utilise `Leave` pour indiquer la sortie de cet accès exclusif.

```
void Enter();  
void Leave();
```

La classe `wxCriticalSectionLocker`

La classe `wxCriticalSectionLocker` est une classe utilitaire pour faciliter l'utilisation de la classe `wxCriticalSection`. Avec cette classe, vous indiquez l'instance de `wxCriticalSection` à utiliser dans le constructeur et vous n'avez plus rien d'autre à faire, pas de fonction `Enter` ou `Leave` à appeler.

```
void SetData()  
{  
    // CritSect_Data est un object wxCriticalSection synchronisant l'accès  
    // à la donnée Data  
    wxCriticalSectionLocker locker(CritSect_Data);  
  
    if ( ... )  
    {  
        // faire quelque chose avec la donnée Data  
        ...  
  
        return;  
    }  
  
    // Faire autre chose avec Data  
    ...  
  
    return;  
}
```

Création de classes dérivées de `wxThread`

Nous allons créer des classes pour effectuer trois tâches :

- Gestion des fichiers éloigné.

- Transfert montant « upload ».
- Transfert descendant « download ».

Pour réaliser ceci nous aurons besoin de cinq classes dont la hiérarchie est exposée sur le graphique ci-dessous.

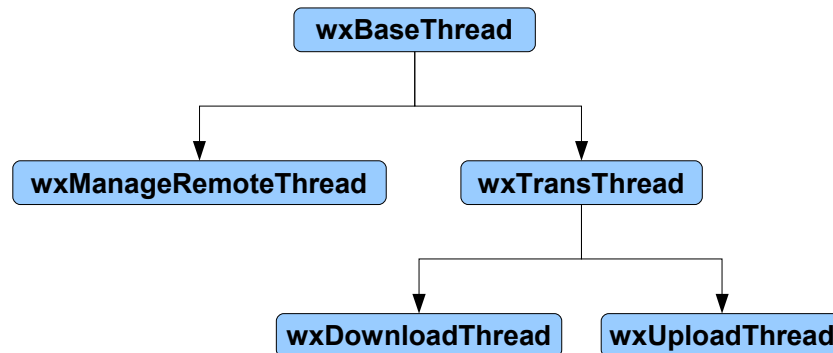


Illustration 22: Hiérarchie des classes de tâches

Avant de commencer le code de ces classes, nous allons légèrement modifier la classe wxMyFTP et la classe wxMyFTPApp. Dans le fichier myftp.h, ajoutez dans la section publique de la classe wxMyFTP la ligne ci-dessous afin d'avoir un objet de synchronisation pour l'accès à l'instance de cette classe.

```
wxCriticalSection m_ftp_critsect;
```

Puis dans le fichier myftppapp.h, ajoutez les lignes suivantes dans la partie publique de la classe wxMyFTPApp

```

wxCriticalSection m_array_ftp_critsect;

    wxCriticalSection m_data_critsect;
    wxString remotePath;

    wxCriticalSection m_array_thread_critsect;
    wxArrayThread m_threads;
    wxSemaphore m_semAllDone;
    bool m_waitingUntilAllDone;

    wxString GetRemotePath();
    bool GetConnected();

    wxOutputStream* GetOutputStream(wxMyFTP* ftp, const wxString& path,
                                     const wxString& filename, bool ascii);
    wxInputStream* GetInputStream(wxMyFTP* ftp, const wxString& path,
                                  const wxString& filename, bool ascii);

    void Using(wxMyFTP* ftp);
    bool Used(wxMyFTP* ftp);

    bool CurrentOnStreaming();
  
```

Et dans la partie private la ligne

```
wxCriticalSection m_used_critsect;
```

Ainsi que les deux lignes suivantes au dessus de la déclaration de la classe wxMyFTPApp.

```
class wxThread;
```

```
WX_DEFINE_ARRAY_PTR(wxThread*, wxArrayThread);
```

Dans le fichier `myftppapp.cpp`, modifiez le constructeur de la classe `wxMyFTPApp` comme indiqué.

```
wxMyFTPApp::wxMyFTPApp() : m_semAllDone()
{
    m_UsedFTP = 0;
    m_waitingUntilAllDone = false;
    bConnected = false;
}
```

Dans l'implémentation de la fonction `OnInit` ajoutez avant l'appel de `OnInit` de la classe de base l'appel à la fonction statique `Initialize` de la classe `wxSocketBase`.

```
wxSocketBase::Initialize();
```

Puis ajoutez l'implémentation des fonctions `GetRemotePath` et `GetConnected`, vous remarquerez l'utilisation de la classe utilitaire `wxCriticalSectionLocker` avec l'instance `m_data_critsect` qui permet de synchroniser l'accès aux données protégées par celle `m_data_critsect`.

```
wxString wxMyFTPApp::GetRemotePath()
{
    wxCriticalSectionLocker locker(m_data_critsect);
    return remotePath;
}

bool wxMyFTPApp::GetConnected()
{
    wxCriticalSectionLocker locker(m_data_critsect);
    return bConnected;
}
```

Ajoutez au début de l'implémentation de la fonction `CreateFTP` la ligne suivante afin de synchroniser l'accès au tableau d'objet FTP.

```
wxCriticalSectionLocker enter(m_array_ftp_critsect);
```

Enfin implémentez les fonctions membre comme indiqué, vous remarquerez l'utilisation des objets de synchronisation respectif pour chaque lecture ou modification d'objet ou de données.

```
wxOutputStream* wxMyFTPApp::GetOutputStream(wxMyFTP* ftp, const wxString& path,
                                             const wxString& filename, bool ascii)
{
    wxCriticalSectionLocker enter(ftp->m_ftp_critsect);

    if (ftp)
        return ftp->GetOutputStream(path, filename, ascii);
    else
        return NULL;
}

wxInputStream* wxMyFTPApp::GetInputStream(wxMyFTP* ftp, const wxString& path,
                                          const wxString& filename, bool ascii)
{
    wxCriticalSectionLocker enter(ftp->m_ftp_critsect);

    if (ftp)
        return ftp->GetInputStream(path, filename, ascii);
    else
        return NULL;
}

void wxMyFTPApp::Using(wxMyFTP* ftp)
{
    wxCriticalSectionLocker enter(m_used_critsect);
```

```

    m_UsedFTP = ftp;
}

bool wxMyFTPApp::Used(wxMyFTP* ftp)
{
    wxCriticalSectionLocker enter(m_used_critsect);

    return (m_UsedFTP == ftp);
}

bool wxMyFTPApp::CurrentOnStreaming()
{
    wxCriticalSectionLocker enter(m_used_critsect);

    if (m_UsedFTP)
    {
        wxCriticalSectionLocker locker(m_UsedFTP->m_ftp_critsect);

        return m_UsedFTP->OnStreaming();
    }
    else
        return false;
}

```

Afin que les objet wxThread puissent accéder à certaine partie de l'interface nous allons ajouter quatre fonctions membres à la fenêtre principale de notre application. Dans la partie publique de la déclaration de la classe wxMyFTPFrame ajoutez les quatre lignes suivantes.

```

wxTextCtrl* GetTextCtrlInfo() {return m_TextCtrlInfo;}
wxListCtrl* GetTransfertList() {return m_TransfertList;}
wxListCtrl* GetLocalFiles() {return m_LocalFiles;}
wxListCtrl* GetRemoteFiles() {return m_RemoteFiles;}

```

La classe wxBaseThread

C'est la classe de base de toutes les classes de tâches de notre application. Elle contient un pointeur sur l'objet fenêtre de l'application et un pointeur sur un objet *wxFTP*. Ces deux données sont fournies en paramètres au constructeur. Cette classe comprend aussi un destructeur, une fonction d'affichage d'information et une fonction de connexion. Créez un fichier *basethread.h* sans l'ajouter au projet et saisissez la déclaration de la classe *wxBaseThread* comme indiqué ci-dessous.

```

#ifndef _BASETHREAD_H_
#define _BASETHREAD_H_

#ifdef __GNUG__ && !defined(__APPLE__)
#pragma interface "basethread.cpp"
#endif

class wxMyFTPFrame;
class wxMyFTP;

class wxBaseThread : public wxThread
{
public:
    wxBaseThread(wxMyFTPFrame* frame, wxMyFTP* ftp);

    virtual ~wxBaseThread();

protected:

```

```
wxMyFTPFrame* m_pFrame;
wxMyFTP* m_pFtp;

void UpdateLog(const wxString& msg);

bool Connecting();

};

#endif // _BASETHREAD_H_
```

Créez maintenant le fichier *basethread.cpp* et ajoutez le au projet, ce fichier contiendra l'implémentation de notre classe *wxBaseThread*. Nous allons maintenant nous intéresser au constructeur et au destructeur de la classe *wxBaseThread*. Le constructeur est très simple: il ne fait que l'affectation des paramètres aux données de la classe. En revanche le destructeur est plus complexe, car il faut gérer la possibilité de demande de destruction de toutes les tâches et aussi si nécessaire de la fermeture et de la suppression de l'instance de *wxMyFTP*. Remarquez l'utilisation suivant les cas des objets *wxCriticalSectionLocker* ou de l'appel aux fonction *Enter* et *Leave* des objets *wxCriticalSection*.

```
#if defined(__GNUG__) && !defined(__APPLE__)
#pragma implementation "basethread.h"
#endif

// For compilers that support precompilation, includes "wx/wx.h".
#include "wx/wxprec.h"

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include <wx/wx.h>
#endif

#include "basethread.h"
#include "myftppapp.h"
#include "myftpframe.h"
#include "myftp.h"

wxBaseThread::wxBaseThread(wxMyFTPFrame* frame, wxMyFTP* ftp) : wxThread()
{
    m_pFrame = frame;
    m_pFtp = ftp;
}

wxBaseThread::~wxBaseThread()
{
    // Accès exclusif au tableau de tâches
    wxCriticalSectionLocker locker(wxGetApp().m_array_thread_critsect);

    wxArrayThread& threads = wxGetApp().m_threads;

    threads.Remove(this); // Suppression dans le tableau

    if (threads.IsEmpty()) // Si le tableau est vide
    {
        if (wxGetApp().m_waitingUntilAllDone) // et que le flag de demande
        { // de fin de toutes les tâches est positionné alors
            wxGetApp().m_waitingUntilAllDone = false; // supprime le flag
            wxGetApp().m_semAllDone.Post(); // et envoi le signal
        }
    }

    m_pFtp->m_ftp_critsect.Enter(); // Accès exclusif à l'objet wxFTP
}
```

```

if (!m_pFtp->OnStreaming() && !wxGetApp().Used(m_pFtp))
{
    // L'objet wxFTP n'est pas en transfert ni utilisé comme répertoire distant
    // Accès exclusif au tableau d'objet wxFTP
    wxCriticalSectionLocker enter(wxGetApp().m_array_ftp_critsect);
    m_pFtp->SetDefaultTimeout(0);
    m_pFtp->Notify(false); // Pour être sûr qu'aucune notification ne sera générée
    wxGetApp().m_ftps.Remove(m_pFtp); // Suppression de l'objet dans le tableau
    m_pFtp->m_ftp_critsect.Leave(); // Sortie du mode exclusif sur wxFTP
    m_pFtp->Destroy(); // Pour toutes les sockets Destroy est préférable à delete
}
else
    m_pFtp->m_ftp_critsect.Leave(); // Sortie du mode exclusif sur wxFTP
}

```

Ajoutez au fichier l'implémentation de la fonction *UpdateLog*, remarquez l'utilisation des fonctions *wxMutexGuiEnter* et *wxMutexGuiLeave* qui encadre la mise à jour de l'interface.

```

void wxBaseThread::UpdateLog(const wxString& msg)
{
    if (!TestDestroy()) // Si aucune demande de destruction
    {
        wxMutexGuiEnter(); // Demande l'accès exclusif à l'interface

        m_pFrame->GetTextCtrlInfo()->AppendText(msg); // Affiche le message
        m_pFrame->GetTextCtrlInfo()->AppendText(_T("\n")); // Ligne suivante

        wxMutexGuiLeave(); // Libère l'accès à l'interface
    }
}

```

L'implémentation de la fonction *Connecting*, permet d'établir une connexion tout en surveillant une demande de destruction de la tâche en attendant que la connexion soit effective.

```

bool wxBaseThread::Connecting()
{
    // Instaure l'accès exclusif à l'objet wxFTP
    wxCriticalSectionLocker locker(m_pFtp->m_ftp_critsect);

    if (m_pFtp->IsDisconnected()) // si déjà connecté inutile de faire le traitement
    {
        int CountLoop = 0;

        UpdateLog(wxString::Format(_("Trying to connect to %s"),
            m_pFtp->Hostname().c_str()));

        m_pFtp->Connect(); // Connexion

        // Boucle en testant si demande de destruction ou connexion effective
        while (m_pFtp->IsDisconnected() && !TestDestroy())
        {
            wxThread::Sleep(100);
            CountLoop++;

            if (CountLoop > 100)
                break;
        }
        if (TestDestroy()) // Si demande de destruction retour faux
        {
            m_pFtp->Close();
            return false;
        }
        if (m_pFtp->IsDisconnected()) // Si pas de connexion
        { // Affichage d'un message et retour faux

```

```
        UpdateLog(wxString::Format(_("Unable to connect to %s"),
                                   m_pFtp->Hostname().c_str()));
        return false;
    }

    return true;
}
```

La classe *wxManageRemoteThread*

Avant de créer la classe *wxManageRemoteThread* nous allons ajouter une petite classe dans le fichier *myftpapp.h* puis modifier légèrement la classe *wxMyFTPApp*. Cette nouvelle classe nous permettra de spécifier quelles actions doit faire la tâche *wxManageRemoteThread*. Saississez les lignes suivantes avant la déclaration de la classe *wxMyFTPApp*.

```
enum wxFtpJobs {mfjNone, mfjRefresh, mfjChangeDir, mfjCreateDir,
               mfjRemoveDir, mfjDelete, mfjRename, mfjDisconnect};

class wxJob
{
public:

    wxJob() : jobsFtp(mfjNone){}
    wxJob(wxFtpJobs action, const wxString& str1, const wxString& str2) :
        jobsFtp(action){jobsString1 = str1; jobsString2 = str2;}

    wxFtpJobs jobsFtp;
    wxString jobsString1;
    wxString jobsString2;
};

WX_DEFINE_ARRAY_PTR(wxJob*, wxArrayJob);
```

Comme vous pouvez le constater cette classe est très simple, elle permet juste de spécifier une action contenue dans l'énumération *wxFtpJobs* et deux chaînes en complément pour indiquer les paramètres de certaines actions comme par exemple *mfjRename*.

Dans la partie publique de la déclaration de la classe *wxMyFTPApp*, ajoutez les deux lignes ci-dessous afin de créer un tableau dynamique de classe *wxJob*.

```
wxCriticalSection m_array_job_critsect;
wxArrayJob m_jobs;
```

Nous allons pouvoir enfin créer notre classe, les deux fonctions membres principales de celle-ci sont *UpdateDisplay* et bien évidemment *Entry* qu'il faut redéfinir pour chaque tâche. Les autres fonctions sont assez simple, et n'ont la plupart du temps que la spécificité d'utiliser un objet de synchronisation.

Créez le fichier *manageremote.h* mais ne l'ajoutez pas au projet, puis saisissez la déclaration de la classe *wxManageRemoteThread*.

```
#ifndef _MANAGEREMOTE_H_
#define _MANAGEREMOTE_H_

#ifdef __GNUG__ && !defined(__APPLE__)
#pragma interface "manageremote.cpp"
```

```

#endif
#include "baseThread.h"

class wxMyFTPFrame;
class wxMyFTP;
class wxJob;

class wxManageRemoteThread : public wxBaseThread
{
public:

    wxManageRemoteThread(wxMyFTPFrame* frame, wxMyFTP* ftp) :
        wxBaseThread(frame, ftp){}

    virtual void* Entry();

private:

    void UpdateDisplay();
    void ClearDisplay();
    void UpdateMenu();
    void UpdatePath();
    void ClearJobs();

    void SetConnect(bool value);

    wxJob* GetNextJob();
};

#endif // _MANAGEREMOTE_H_

```

La structure de la fonction membre *Entry* est une boucle qui s'effectue tant que l'on est connecté ou qu'aucune demande de destruction ne soit demandée. A l'intérieur de cette boucle le prochain travail à effectuer est extrait du tableau *m_jobs* de la classe application et une instruction *switch* permet d'accomplir la tâche définie. Créez un fichier *manageremote.cpp* et ajoutez le au projet, puis implémentez les fonctions membres de la classe *wxManageRemoteThread*.

```

#ifdef __GNUC__ && !defined(__APPLE__)
#pragma implementation "manageremote.h"
#endif

// For compilers that support precompilation, includes "wx/wx.h".
#include <wx/wxprec.h>

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifdef WX_PRECOMP
#include <wx/wx.h>
#endif

#include <wx/dirctrl.h>
#include <wx/datetime.h>
#include "manageremote.h"
#include "myftpapp.h"
#include "myftpframe.h"
#include "myftp.h"

void* wxManageRemoteThread::Entry()
{
    wxJob* job;
    bool bRet, bAutoClose = true;

    wxGetApp().Using(m_pFtp); // Activation du flag d'utilisation

```

```
if (Connecting()) // Tentative de connexion
{
    UpdateLog(wxString::Format(_("Connected to %s"), m_pFtp->Hostname().c_str()));
    SetConnect(true);
}
else
{
    wxGetApp().Using(0); // Désactivation du flag d'utilisation
    return NULL;
}
UpdateMenu();
UpdateDisplay();
UpdateLog(m_pFtp->GetLastResult());
UpdatePath();

// On boucle tant que la connexion est active et qu'aucune demande de destruction
while (m_pFtp->IsConnected() && !TestDestroy())
{
    job = GetNextJob(); // Récupération de travail

    if ((job == NULL) || (job->jobsFtp == mfjNone))
        wxThread::Sleep(100); // Aucun on attend un peu 1/10e de seconde
    else
        switch (job->jobsFtp) // Dispatche le travail
        {
            case mfjNone: // pour éviter un warning
                break;
            case mfjRefresh :
                UpdateDisplay();
                UpdatePath();
                break;
            case mfjChangeDir :
                if (job->jobsString1 == _T("..")) // Si demande répertoire parent
                    bRet = m_pFtp->CdUp(); // Utilisation de la nouvelle commande
                else
                    bRet = m_pFtp->ChDir(job->jobsString1);
                if (bRet)
                {
                    UpdateDisplay();
                    UpdatePath();
                }
                break;
            case mfjCreateDir :
                bRet = m_pFtp->MkDir(job->jobsString1);
                UpdateLog(m_pFtp->GetLastResult());
                if (bRet)
                {
                    UpdateDisplay();
                    UpdatePath();
                }
                break;
            case mfjRemoveDir :
                m_pFtp->RmDir(job->jobsString1);
                UpdateLog(m_pFtp->GetLastResult());
                break;
            case mfjDelete :
                m_pFtp->RmFile(job->jobsString1);
                UpdateLog(m_pFtp->GetLastResult());
                break;
            case mfjRename :
                bRet = m_pFtp->Rename(job->jobsString1, job->jobsString2);
                UpdateLog(m_pFtp->GetLastResult());
                if (bRet)
                {
                    UpdateDisplay();
                    UpdatePath();
                }
                break;
            case mfjDisconnect : // demande de déconnexion de l'utilisateur
                ClearJobs(); // Effacement de la file de travaux
```

```

        delete job;          // Suppression du travail
        bAutoClose = false;
        SetConnect(false);
        UpdateMenu();
        ClearDisplay();      // Effacement de la liste
        UpdateLog(wxString::Format(_("Disconnected from %s"), ↓
m_pFtp->Hostname().c_str()));
        wxGetApp().Using(0); // Désactivation du flag d'utilisation
        return NULL;        // Sortie de la tâche
    }
    if (job != NULL)
        delete job; // Suppression du travail
}
if (bAutoClose&&!TestDestroy())
{ // Fermeture automatique
    SetConnect(false);
    UpdateMenu();
    ClearDisplay();
    UpdateLog(wxString::Format(_("Disconnected from %s (timeout)"), m_pFtp-
>Hostname().c_str()));
}
ClearJobs();          // Effacement de la file de travaux
wxGetApp().Using(0); // Désactivation du flag d'utilisation
return NULL;
}

```

La fonction `UpdateDisplay` récupère les noms de fichiers avec d'autres informations et découpe les chaînes reçues. Le découpage utilisé est le plus courant rencontré sur la plupart des serveur FTP, cependant certains serveurs peuvent renvoyer des informations différentes ou dans un ordre différent. Il faudra dans ce cas vérifier le type de serveur avec la commande FTP « SYST » et modifier cette fonction afin qu'elle gère différents types de serveurs.

```

void wxManageRemoteThread::UpdateDisplay()
{
    wxArrayString files, fTmp;
    wxListCtrl* remote;
    long m_count;
    int image_id;
    bool bRepParent = false;
    bool IsCnx;

    wxMutexGuiEnter(); // Utilisation exclusive de la GUI

    remote = (wxListCtrl*)m_pFrame->GetRemoteFiles();
    remote->DeleteAllItems(); // Effacement de la liste

    wxMutexGuiLeave(); // Fin de l'utilisation exclusive

    m_pFtp->m_ftp_critsect.Enter();
    IsCnx = m_pFtp->IsConnected(); // Vérification de la connexion
    m_pFtp->m_ftp_critsect.Leave();
    if (IsCnx)
    {
        wxString dirItems, tmp, name, type, size, date, attr;
        size_t count;
        bool bRet;

        m_pFtp->m_ftp_critsect.Enter();
        bRet = m_pFtp->GetDirList(files); // récupération de la liste de fichiers
        m_pFtp->m_ftp_critsect.Leave();

        if (!bRet)
            UpdateLog(_("Problem to receive files list!"));
        else
        {

```

```
count = files.GetCount();
for (size_t i = 0; i < count; i++)
{
    dirItems = files.Item(i);
    // Exemple de chaînes reçues
    // drwxr-xr-x 1 ftp ftp          0 Sep 05 19:27 bin
    // -r--r--r-- 1 guilhem lavaux    12738 Jan 16 20:17 cmndata.cpp
    // Découpage de chaque chaîne
    attr = dirItems.BeforeFirst(' ');
    tmp = dirItems.AfterFirst(' ').Strip(wxString::leading);
    type = tmp.BeforeFirst(' ');
    tmp = tmp.AfterFirst(' ').Strip(wxString::leading);

    tmp = tmp.AfterFirst(' ').Strip(wxString::leading);
    tmp = tmp.AfterFirst(' ').Strip(wxString::leading);

    size = tmp.BeforeFirst(' ');
    tmp = tmp.AfterFirst(' ').Strip(wxString::leading);
    date = tmp.Left(12);
    name = tmp.Mid(13);
    fTmp.Add(name);
    if (name != _T(".")) // Exclure le lien vers le répertoire lui même
    {
        if (name == _T("..")) // On vérifie l'existence d'un retour
            bRepParent = true; // au répertoire parent
        wxMutexGuiEnter(); // Utilisation exclusive de la GUI
        m_count = remote->GetItemCount();
        m_count = remote->InsertItem(m_count, name); // Insertion de l'élément
        if (attr.substr(0, 1) == _T("d")) // est-ce un répertoire?
        { // on indique que c'est un répertoire au lieu de la taille
            remote->SetItem(m_count, 1, _T("<DIR>"));
            // Utilisation de wxFileIconsTable::folder pour l'image de répertoire
            remote->SetItemImage(m_count, wxFileIconsTable::folder);
            // On indique que c'est un répertoire en positionnant un index négatif
            remote->SetItemData(m_count, -(i + 1));
        }
        else
        {
            remote->SetItem(m_count, 1, size); // On indique la taille
            image_id = wxFileIconsTable::file; // Image par défaut pour un fichier
            if (name.Find(_T('.')) != wxNOT_FOUND) // Si possible on cherche l'icône
                image_id = wxTheFileIconsTable->GetIconID(name.AfterLast(_T('.')));
            remote->SetItemImage(m_count, image_id);
            // on indique que c'est un fichier en positionnant un index positif
            remote->SetItemData(m_count, i + 1);
        }
        wxDateTime dateTime;
        dateTime.ParseDate(date.c_str());
        tmp = date.AfterLast(' ');
        date = dateTime.FormatISODate() + _T(" ");
        if (tmp.Find(':') != -1)
            date += tmp;
        else // Si l'heure n'est pas indiquée mettre 12:00 par défaut
            date += _T("12:00");
        remote->SetItem(m_count, 2, date); // On indique la date et l'heure
        wxMutexGuiLeave(); // Fin de l'utilisation exclusive
    }
}
if (bRepParent == false)
{ // Il n'y a pas de retour au parent on l'ajoute
    wxMutexGuiEnter();
    name = _T("..");
    fTmp.Add(name);
    count = remote->GetItemCount();
    m_count = remote->InsertItem(count, name);
    remote->SetItem(m_count, 1, _T("<DIR>"));
    remote->SetItemImage(m_count, wxFileIconsTable::folder);
    remote->SetItemData(m_count, -(count + 1));
    wxDateTime dateTime = wxDateTime::Now();
}
```

correspondante

```

        date = dateTime.FormatISODate() + _T(" ") + dateTime.FormatISOTime().Left(5);
        remote->SetItem(m_count, 2, date);
        wxMutexGuiLeave();
    }
    wxMutexGuiEnter();
    // Tri par ordre alpha avec les répertoires d'abord
    // Pour cela, on se sert de la liste de nom que l'on fourni en paramètre
    // et des index positionnés par SetItemData négatif pour les répertoires
    // et positif pour les fichiers
    remote->SortItems(CompareList, (long)&fTmp);
    wxMutexGuiLeave();
}
}
}

```

Les autres fonctions membres ont assez peu d'intérêt au niveau de la programmation. Notez juste l'utilisation des fonctions *wxMutexGuiEnter* et *wxMutexGuiLeave* pour avoir un accès exclusif à l'interface, et à la classe utilitaire *wxCriticalSectionLocker* pour synchroniser l'accès à certaines données.

```

void wxManageRemoteThread::ClearDisplay()
{
    wxListCtrl* remote;

    wxMutexGuiEnter();

    remote = (wxListCtrl*)m_pFrame->GetRemoteFiles();
    remote->DeleteAllItems();

    wxMutexGuiLeave();
}

void wxManageRemoteThread::SetConnect(bool value)
{
    wxCriticalSectionLocker locker(wxGetApp().m_data_critsect);
    wxGetApp().bConnected = value;
}

void wxManageRemoteThread::UpdateMenu()
{
    wxMutexGuiEnter();
    m_pFrame->EnableMenu();
    wxMutexGuiLeave();
}

void wxManageRemoteThread::UpdatePath()
{
    wxString path = m_pFtp->Pwd();
    wxCriticalSectionLocker locker(wxGetApp().m_data_critsect);

    wxGetApp().remotePath = path;
}

wxJob* wxManageRemoteThread::GetNextJob()
{
    wxJob* job = NULL;

    wxCriticalSectionLocker locker(wxGetApp().m_array_job_critsect);

    if (wxGetApp().m_jobs.GetCount())
    {
        job = wxGetApp().m_jobs[0];
        wxGetApp().m_jobs.Remove(job);
    }
    return job;
}

```

```
void wxManageRemoteThread::ClearJobs()
{
    wxJob* job;
    wxCriticalSectionLocker locker(wxGetApp().m_array_job_critsect);

    size_t count = wxGetApp().m_jobs.GetCount();

    if (count)
    {
        while (!wxGetApp().m_jobs.IsEmpty())
        {
            job = wxGetApp().m_jobs.Last();
            wxGetApp().m_jobs.Remove(job);
            delete job;
        }
    }
}
```

La classe wxTransThread

C'est la base des classes de transfert. Les données nécessaires pour effectuer le transfert sont stockées dans cette classe, les fonctions permettant la mise à jour de l'interface sont aussi définies dans celle-ci.

Les variables `m_AsciiMode`, `m_LocalPath`, `m_RemotePath` et `m_FileName` permettent de stocker les valeurs fournies par le constructeur. Les variables `m_FileSize`, `m_SizeRead` et `m_Buffer` sont utilisées pendant le transfert. Les fonctions `UpdateTransfertItemStatus`, `UpdateTransfertItem`, `DeleteTransfertItem` et `GetTransfertItemIndex` servent à la mise à jour de l'interface.

Nous allons définir la classe *wxTransThread* dans le fichier *transthread.h*, que nous allons créer sans l'insérer dans le projet.

```
#ifndef _TRANSTHREAD_H_
#define _TRANSTHREAD_H_

#if defined(__GNUG__) && !defined(__APPLE__)
#pragma interface "transthread.cpp"
#endif
#include "basethread.h"

#define BUFFSIZE 8192

enum wxUpdateMode {umStarting, umWaiting, umTransferring, umFinishing};

class wxTransThread : public wxBaseThread
{
public:
    wxTransThread(wxMyFTPFrame* frame, wxMyFTP* ftp, bool asciimode,
                  const wxString& localpath, const wxString& remotepath,
                  const wxString& filename);

    virtual ~wxTransThread();

protected:
    bool m_AsciiMode;           // mode de récupération Ascii/Binary
    wxString m_LocalPath;       // Chemin local
    wxString m_RemotePath;       // Chemin distant
    wxString m_FileName;        // Nom du fichier
    size_t m_FileSize;          // Taille du fichier
    size_t m_SizeRead;          // Nombre d'octets lus
```

```

char m_Buffer[BUFFSIZE];    // Buffer de transfert

// Mise à jour du statut de la tâche dans le wxListCtrl
void UpdateTransfertItemStatus(wxUpdateMode mode);
// Mise à jour de la progression de la tâche dans le wxListCtrl
void UpdateTransfertItem();
// Suppression de l'item dans le wxListCtrl
void DeleteTransfertItem();

private:

// Retrouve l'index de l'item dans le wxListCtrl
long GetTransfertItemIndex();
};

#endif    // _TRANSTHREAD_H_

```

Avant de pouvoir implémenter la classe *wxTransThread*, nous allons créer une autre petite classe afin de conserver dans la liste des transferts à exécuter les données pour effectuer ces transferts. Cette nouvelle classe nommée *wxTransJob* sera déclarée dans le fichier *myftpframe.h* avant la déclaration de la classe *wxMyFTPFrame*.

```

#define ID_CMD_LAUNCHTHREAD    10701

class wxTransThread;

enum wxTransfertType {ttUpload, ttDownload};

class wxTransJob
{
public:

    wxTransJob(const wxString& remote, const wxString& local,
               const wxString& file, wxTransfertType type,
               const wxString& hostname, unsigned short service,
               const wxString& user, const wxString& password,
               int mode, bool passive)
    {
        m_RemotePath = remote;
        m_LocalPath = local;
        m_Filename = file;
        m_Type = type;
        m_Hostname = hostname;
        m_Service = service;
        m_User = user;
        m_Password = password;
        m_TransfertMode = mode;
        m_Passive = passive;
        m_Thread = 0;
    }

    wxString m_RemotePath;
    wxString m_LocalPath;
    wxString m_Filename;
    wxTransfertType m_Type;

    wxString m_Hostname;
    unsigned short m_Service;
    wxString m_User;
    wxString m_Password;
    int m_TransfertMode;
    bool m_Passive;

    wxTransThread* m_Thread;
};

```

Nous commencerons par le constructeur et le destructeur de la classe `wxTransThread`. Le constructeur n'a rien de spécial, il ne fait que l'appel du constructeur de la classe parent et l'affectation des paramètres aux données correspondantes de la classes. Le destructeur quant à lui nous montre que nous pouvons envoyer des messages depuis une tâche vers la fenêtre principale de l'application. Cela est assez pratique pour indiquer la fin de certaines actions au sein d'une tâche afin de synchroniser d'autres traitements.

```
#if defined(__GNUG__) && !defined(__APPLE__)
#pragma implementation "transthread.h"
#endif

#include <wx/wxprec.h>

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include <wx/wx.h>
#endif

#include "transthread.h"
#include "myftpframe.h"

wxTransThread::wxTransThread(wxMyFTPFrame* frame, wxMyFTP* ftp, bool asciimode,
                             const wxString& localpath, const wxString& remotepath,
                             const wxString& filename) :
    wxBaseThread(frame, ftp)
{
    m_AsciiMode = asciimode;
    m_LocalPath = localpath;
    m_RemotePath = remotepath;
    m_FileName = filename;
    m_FileSize = m_SizeRead = 0;
}

wxTransThread::~wxTransThread()
{
    // Avant de sortir envoyer un message de lancement de nouveau transfert
    wxCommandEvent event(wxEVT_COMMAND_MENU_SELECTED, ID_CMD_LAUNCHTHREAD);
    wxPostEvent(m_pFrame, event);
}
```

Les fonctions suivantes permettent la mise à jour de l'interface et plus particulièrement de la liste des tâches de transfert. Vous remarquerez que seule `GetTransfertItemIndex` est privée et n'utilise pas de fonctions pour l'utilisation exclusive de l'interface, ceci est fait car cette fonction est appelée par les autres fonctions qui, elles utilisent les fonctions `wxMutexGuiEnter` et `wxMutexGuiLeave`.

```
void wxTransThread::UpdateTansfertItemStatus(wxUpdateMode mode)
{
    if (!TestDestroy()) // Si aucune demande de destruction
    {
        long lCount;
        wxString sStatus;

        wxMutexGuiEnter(); // Utilisation exclusive de la GUI

        lCount = GetTransfertItemIndex(); // Récupère l'index
        if (lCount != -1)
        { // Préparation de la chaîne de statut
            switch (mode)
```

```

        {
            case umStarting :
                sStatus = _("Starting");
                break;
            case umWaiting :
                sStatus = _("Waiting");
                break;
            case umTransferring :
                sStatus = _("Transferring");
                break;
            default :
                sStatus = _("Finishing");
        }
        // Affichage de la chaine
        m_pFrame->GetTransfertList()->SetItem(lCount, 4, sStatus);
    }

    wxMutexGuiLeave();        // Fin de l'utilisation exclusive de la GUI
}

void wxTransThread::UpdateTransfertItem()
{
    if (!TestDestroy())      // Si aucune demande de destruction
    {
        long lCount;
        wxString sNbBytes, sPCent;

        wxMutexGuiEnter();    // Utilisation exclusive de la GUI

        lCount = GetTransfertItemIndex();    // Récupère l'index
        if (lCount != -1)
        {
            // Affiche le nombre d'octets lus
            sNbBytes = wxString::Format(_T("%u"), m_SizeRead);
            m_pFrame->GetTransfertList()->SetItem(lCount, 3, sNbBytes);
            // Si une taille de fichier a été trouvée
            if (m_FileSize != 0)
            {
                // Calculer le pourcentage lu et l'afficher
                double dPCent = (double)m_SizeRead * 100.0 / (double)m_FileSize;
                if (dPCent > 100.0)
                    dPCent = 100.0;
                sPCent = wxString::Format(_T("%u%%"), (size_t)dPCent);
            }
            else // Taille non trouvée
                sPCent = _T("N/A");
            m_pFrame->GetTransfertList()->SetItem(lCount, 5, sPCent);
        }

        wxMutexGuiLeave();    // Fin de l'utilisation exclusive de la GUI
    }
}

void wxTransThread::DeleteTransfertItem()
{
    if (!TestDestroy())      // Si aucune demande de destruction
    {
        long lCount;

        wxMutexGuiEnter();    // Utilisation exclusive de la GUI

        lCount = GetTransfertItemIndex();    // Récupère l'index
        if (lCount != -1) // Supprime l'item à l'index trouvé
            m_pFrame->GetTransfertList()->DeleteItem(lCount);

        wxMutexGuiLeave();    // Fin de l'utilisation exclusive de la GUI
    }
}

long wxTransThread::GetTransfertItemIndex()

```

```
{
    long lCount, i;
    wxTransJob* TransJob;

    lCount = m_pFrame->GetTransfertList()->GetItemCount();
    // Récupère l'index de l'item qui à son data (wxTransJob*) dont la donnée m_Thread
    // est identique au pointeur de la tâche
    for (i = 0; i < lCount; i++)
    {
        TransJob = (wxTransJob*)m_pFrame->GetTransfertList()->GetItemData(i);
        if ((TransJob != 0)&&(TransJob->m_Thread == this))
            return i;
    }
    return -1;
}
```

La classe wxDownloadThread

Cette tâche permet la gestion des flux descendants, c'est à dire la récupération de fichiers depuis un serveur FTP. Cette classe est issue de la classe *wxTransThread* qui est décrite précédemment, de ce fait à part le constructeur cette nouvelle classe ne comporte qu'une seule fonction membre qui est *Entry*.

Pour saisir la déclaration de la classe *wxDownloadThread* vous allez créer un nouveau fichier nommé *downloadthread.h* que vous n'ajouterez pas au projet.

```
#ifndef _DOWNLOADTHREAD_H_
#define _DOWNLOADTHREAD_H_

#if defined(__GNUG__) && !defined(__APPLE__)
#pragma interface "downloadthread.cpp"
#endif
#include "transthread.h"

class wxDownloadThread : public wxTransThread
{
public:

    wxDownloadThread(wxMyFTPFrame* frame, wxMyFTP* ftp, bool asciimode,
                    const wxString& localpath, const wxString& remotepath,
                    const wxString& filename) :
        wxTransThread(frame, ftp, asciimode, localpath, remotepath, filename){}

    virtual void* Entry();
};

#endif // _DOWNLOADTHREAD_H_
```

Vous allez à présent créer un fichier *downloadthread.cpp* que vous ajouterez au projet afin d'implémenter la fonction *Entry* de la classe *wxDownloadThread*. Cette fonction se divise en quatre parties :

- Récupération de la taille approximative du fichier d'entrée.
- Création des flux d'entrée et de sortie.
- Boucle de lecture du flux d'entrée et d'écriture du flux de sortie.
- Suppression des flux d'entrée et de sortie.

```

#if defined(__GNU__) && !defined(__APPLE__)
#pragma implementation "downloadthread.h"
#endif

#include <wx/wxprec.h>

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include <wx/wx.h>
#endif

#include <wx/wfstream.h>
#include <wx/filename.h>
#include "downloadthread.h"
#include "myftpapp.h"
#include "myftpframe.h"
#include "myftp.h"

void* wxDownloadThread::Entry()
{
    bool ret;
    size_t countread;
    wxFileName fileName;
    wxFileOutputStream* out = NULL;
    wxInputStream* in = NULL;

    UpdateTansfertItemStatus(umStarting);

    {
        wxCriticalSectionLocker locker(m_pFtp->m_ftp_critsect);
        // récupération de la taille du fichier si possible
        int s = m_pFtp->GetFileSize(m_RemotePath, m_FileName);
        if (s != -1) // Voir si la taille est retournée
            m_FileSize = (size_t)s;
    }
    // Création du flux d'entrée
    in = wxGetApp().GetInputStream(m_pFtp, m_RemotePath, m_FileName, m_AsciiMode);
    if (!in)
    {
        UpdateLog(wxString::Format(_("Coudln't get file %s from %s"),
                                   m_FileName.c_str(), m_pFtp->Hostname().c_str()));
    }
    else
    {
        // Création du chemin global local
        fileName.AssignDir(m_LocalPath);
        fileName.SetFullName(m_FileName);
        // Création du flux de sortie
        out = new wxFileOutputStream(fileName.GetFullPath());
        if (!out)
            UpdateLog(wxString::Format(_("Unable to create %s"),
                                       fileName.GetFullPath().c_str()));
    }

    {
        wxCriticalSectionLocker locker(m_pFtp->m_ftp_critsect);
        UpdateLog(m_pFtp->GetLastOpenStreamResult());
    }

    if (in && out)
    {
        UpdateTansfertItemStatus(umTransferring); // Mise à jour interface

        while (!TestDestroy())
        {

```

```
ret = !in->Read(m_Buffer, BUFFSIZE); // Lire le flux
countread = in->LastRead(); // Nombre d'octets lus
if (countread)
{ // Si quelque chose est lu, écrire le buffer
out->Write(m_Buffer, countread);
m_SizeRead += countread; // Incrémenter le compteur
UpdateTransfertItem(); // Mettre à jour l'interface
}

if (ret) // Problème de lecture ou fin
break; // Sortir de la boucle
wxThread::Sleep(100);
}

UpdateTransfertItemStatus(umFinishing); // Mise à jour interface
UpdateLog(wxString::Format(_("%s downloaded (%u bytes)"),
m_FileName.c_str(), m_SizeRead));
}

// Suppression des flux
if (in)
delete in;
if (out)
delete out;
DeleteTransfertItem(); // Mettre à jour l'interface

return NULL;
}
```

La classe wxUploadThread

La classe *wxUploadThread* permet de gérer le flux montant, c'est à dire le fait d'envoyer des fichiers vers un serveur FTP. Comme pour la classe *wxDownloadThread* seule la fonction membre *Entry* est déclarée et implémentée.

Créez le fichier *uploadthread.h* sans l'ajouter au projet, afin de saisir la déclaration de la classe *wxUploadThread*.

```
#ifndef _UPLOADTHREAD_H_
#define _UPLOADTHREAD_H_

#if defined(__GNUG__) && !defined(__APPLE__)
#pragma interface "uploadthread.cpp"
#endif
#include "transthread.h"

class wxUploadThread : public wxTransThread
{
public:

wxUploadThread(wxMyFTPFrame* frame, wxMyFTP* ftp, bool asciimode,
const wxString& localpath, const wxString& remotepath,
const wxString& filename) :
wxTransThread(frame, ftp, asciimode, localpath, remotepath, filename){}

virtual void* Entry();
};

#endif // _UPLOADTHREAD_H_
```

L'implémentation de la fonction membre *Entry* de la classe *wxUploadThread* et presque identique à celle de la classe *wxDownloadThread*, seule la partie pour la

récupération de la taille du fichier d'entrée est réduite et intégrée à la création du flux d'entrée.

Ci-dessous le code du fichier *uploadthread.cpp* que vous allez créer et ajouter au projet.

```

#if defined(__GNUG__) && !defined(__APPLE__)
#pragma implementation "uploadthread.h"
#endif

#include <wx/wxprec.h>

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include <wx/wx.h>
#endif

#include <wx/wfstream.h>
#include <wx/filename.h>
#include "uploadthread.h"
#include "myftpframe.h"
#include "myftp.h"

void* wxUploadThread::Entry()
{
    bool ret;
    size_t countread;
    wxFileName fileName;
    wxOutputStream* out = NULL;
    wxFileInputStream* in = NULL;

    UpdateTansfertItemStatus(umStarting);

    // Création du chemin global local
    fileName.AssignDir(m_LocalPath);
    fileName.SetFullName(m_FileName);
    // Création du flux de sortie
    out = wxGetApp().GetOutputStream(m_pFtp, m_RemotePath, m_FileName, m_AsciiMode);
    if (!out)
    {
        wxCriticalSectionLocker locker(m_pFtp->m_ftp_critsect);

        UpdateLog(wxString::Format(_("Unable to create %s to %s"),
                                    m_FileName.c_str(),
                                    m_pFtp->Hostname().c_str()));
    }
    else
    {
        // Création du flux d'entrée
        in = new wxFileInputStream(fileName.GetFullPath());
        if (!in)
            UpdateLog(wxString::Format(_("Coudln't get file %s"),
                                        fileName.GetFullPath().c_str()));
        else
            m_FileSize = in->GetSize(); // récupération de la taille du fichier
    }

    {
        wxCriticalSectionLocker locker(m_pFtp->m_ftp_critsect);
        UpdateLog(m_pFtp->GetLastOpenStreamResult());
    }

    if (in && out)
    {
        UpdateTansfertItemStatus(umTransferring); // Mise à jour interface

        while (!TestDestroy())

```

```
{
    ret = !in->Read(m_Buffer, BUFFSIZE);
    countread = in->LastRead();
    if (countread)
    { // Si quelque chose est lu, écrire le buffer
        out->Write(m_Buffer, countread);
        m_SizeRead += countread; // Incrémenter le compteur
        UpdateTransfertItem();    // Mettre à jour l'interface
    }

    if (ret) // Problème de lecture ou fin
        break; // Sortir de la boucle
    wxThread::Sleep(100);
}

UpdateTransfertItemStatus(umFinishing); // Mise à jour interface
UpdateLog(wxString::Format(_("%s uploaded (%u bytes)"),
                             m_FileName.c_str(), m_SizeRead));
}

// Suppression des flux
if (in)
    delete in;
if (out)
    delete out;
DeleteTransfertItem(); // Mettre à jour l'interface

return NULL;
}
```

Intégration des classes de tâches

Intégration de la tâche *wxManageRemoteThread*

Nous allons commencer par modifier le fichier *myftpframe.h* en ajoutant l'inclusion du fichier *myftpapp.h*, puis dans la partie privée de la classe *wxMyFTPFrame* ajoutons une donnée booléenne nommée *m_Passive*, enfin nous allons déclarer une nouvelle fonction *SetRemoteAction*.

```
bool m_Passive;

void SetRemoteAction(wxFtpJobs action,
                    const wxString& str1 = wxEmptyString,
                    const wxString& str2 = wxEmptyString);
```

Puis dans le fichier *myftpframe.cpp* nous commencerons par insérer les lignes d'inclusions suivantes afin que nous puissions utiliser les nouvelles classes.

```
#include <wx/filename.h>
#include "myftp.h"
#include "manageremote.h"
```

Nous continuerons par l'implémentation du constructeur de la classe *wxMyFTPFrame* qui jusqu'à présent était vide. Celui ci maintenant devra demander la destruction des tâches en cours avant la fin de l'application et effectuera la destruction des instances d'objet *wxTransJob* associées aux éléments de la liste des tâches.

```
wxMyFTPFrame::~wxMyFTPFrame()
```

```

{
    wxThread *thread;
    wxTransJob* transjob;
    size_t count, i;
    // demande d'accès exclusif au tableau de tâches
    wxGetApp().m_array_thread_critsect.Enter();

    const wxArrayThread& threads = wxGetApp().m_threads;
    count = threads.GetCount();

    if (count)
    {
        wxGetApp().m_waitingUntilAllDone = true;

        while (!threads.IsEmpty()) // Tant que le tableau n'est pas vide
        {
            thread = threads.Last(); // Récupérer un pointeur sur la dernière tâche
            wxGetApp().m_array_thread_critsect.Leave(); // Libérer le tableau de tâche
            thread->Delete(); // demande de destruction de la tâche
            wxGetApp().m_array_thread_critsect.Enter(); // Reprise du tableau de tâche
        }

        wxGetApp().m_array_thread_critsect.Leave(); // Libération du tableau de tâche

        if (count)
            wxGetApp().m_semAllDone.Wait(); // Attendre que le semaphore soit positif

        // Destruction de toutes les instances restantes de wxTransJob dans la liste
        count = m_TransfertList->GetItemCount();
        for (i = 0; i < count; i++)
        {
            transjob = (wxTransJob*)m_TransfertList->GetItemData(i);
            if (transjob != 0)
                delete transjob;
        }
    }
}

```

Ajouter à présent, dans l'implémentation de la fonction *Create* membre de *wxMyFTPFrame*, l'affectation de la donnée *m_Passive*.

```

m_Passive = false;

```

Maintenant complétons l'implémentation de la fonction *OnMnuConnectingClick* qui pour l'instant se contente d'afficher la boîte de dialogue de connexion. Le code que nous ajouterons juste après l'affectation de la variable *m_Service* va permettre la création d'une instance de l'objet *wxManageRemoteThread* et le lancement de la tâche.

```

wxMyFTP* MyFTP;
wxManageRemoteThread* thread;

// recherche ou création du client ftp
MyFTP = wxGetApp().CreateFTP(m_HostName, m_Service, m_User, m_Password, m_Passive);

// Création de l'instance du thread principal
thread = new wxManageRemoteThread(this, MyFTP);
if (thread)
{
    // Creation de la tâche avec un peu de mémoire (4K)
    if (thread->Create(4096) != wxTHREAD_NO_ERROR)
    {
        m_TextCtrlInfo->AppendText(_("Can't create thread!\n"));
        thread->Delete();
    }
    else
    {
        // Ajout de l'instance dans le tableau de tâches
    }
}

```

```
        wxCriticalSectionLocker enter(wxGetApp().m_array_thread_critsect);
        wxGetApp().m_threads.Add(thread);
        // Lancement de la tâche
        if (thread->Run() != wxTHREAD_NO_ERROR)
        {
            m_TextCtrlInfo->AppendText(_("Can't start thread!\n"));
            thread->Delete();
        }
    }
}
else
{
    // L'instance n'a pas pu être créée, on supprime si possible
    // l'instance de wxMyFTP associée
    MyFTP->m_ftp_critsect.Enter();
    if (!MyFTP->OnStreaming() && !wxGetApp().Used(MyFTP))
    {
        wxCriticalSectionLocker enter(wxGetApp().m_array_ftp_critsect);
        wxGetApp().m_ftps.Remove(MyFTP);
        MyFTP->m_ftp_critsect.Leave();
        delete MyFTP;
    }
    else
        MyFTP->m_ftp_critsect.Leave();
}
```

A présent, créons l'implémentation de la fonction `SetRemoteAction`, celle-ci permet d'insérer les actions demandées dans le tableau afin que la tâche `wxManageRemoteThread` puisse les extraire les unes après les autres.

```
void wxMyFTPFrame::SetRemoteAction(wxFtpJobs action, const wxString& str1,
                                   const wxString& str2)
{
    wxJob* job;
    // Vérification qu'aucun transfert n'est en cours
    if (!wxGetApp().CurrentOnStreaming())
    {
        // Blocage de l'accès exclusif au tableau de travaux
        wxCriticalSectionLocker enter(wxGetApp().m_array_job_critsect);

        job = new wxJob(action, str1, str2); // Création d'une instance de wxJob
        wxGetApp().m_jobs.Add(job); // Ajout du travail dans la file d'attente
    }
    else
        m_TextCtrlInfo->AppendText(_("Can't execute command during transfer!\n"));
}
```

Pour terminer nous allons compléter l'implémentation des gestionnaires d'événements suivant :

```
void wxMyFTPFrame::OnMnuDisconnectingClick(wxCommandEvent& event)
{
    SetRemoteAction(mfjDisconnect);
    event.Skip();
}

void wxMyFTPFrame::OnMnuCreateRemoteClick(wxCommandEvent& event)
{
    wxString tmp = wxGetTextFromUser(_("Enter new directory name :"),
                                     _("Create a new remote directory"),
                                     wxEmptyString, this);

    if (!tmp.IsEmpty())
        SetRemoteAction(mfjCreateDir, tmp);
    event.Skip();
}

void wxMyFTPFrame::OnMnuDelRemoteClick(wxCommandEvent& event)
```

```

{
    long item = -1;
    long data;
    wxString tmp;

    do
    {
        // Récupérer l'index du prochain item sélectionné
        item = m_RemoteFiles->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);

        if (item != -1)
        {
            // Récupération du nom et de la donnée
            tmp = m_RemoteFiles->GetItemText(item);
            data = m_RemoteFiles->GetItemData(item);
            if (data < 0) // Vérification si fichier ou répertoire
                SetRemoteAction(mfjRemoveDir, tmp); // en fonction du signe de la donnée
            else
                SetRemoteAction(mfjDelete, tmp);
        }
    }
    while (item != -1); // Boucle tant qu'un élément est trouvé

    SetRemoteAction(mfjRefresh); // Demande de rafraichissement de la liste
    event.Skip();
}

void wxMyFTPFrame::OnMnuRenameClick(wxCommandEvent& event)
{
    wxString tmp1, tmp2;
    long item = -1;
    // Récupérer le premier élément sélectionné
    item = m_RemoteFiles->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);

    if (item != -1)
    {
        // Récupération du nom de l'élément et demande un autre nom à l'utilisateur
        tmp1 = m_RemoteFiles->GetItemText(item);
        tmp2 = wxGetTextFromUser(_("Enter new name :"),
                                _("Rename a remote file/directory"),
                                tmp1, this);

        if (tmp1 != tmp2)
        {
            // Demande de changement de nom puis de rafraichissement
            SetRemoteAction(mfjRename, tmp1, tmp2);
            SetRemoteAction(mfjRefresh);
        }
    }
    event.Skip();
}

void wxMyFTPFrame::OnMnuRefreshRemoteClick(wxCommandEvent& event)
{
    // Demande de rafraichissement
    SetRemoteAction(mfjRefresh);
    event.Skip();
}

```

A ce stade vous pouvez lancer la reconstruction de votre projet avec la combinaison de touche **[CTRL]+[F11]**, puis une fois ce travail achevé vous pouvez lancer votre application avec la combinaison **[CTRL]+[F10]**. Dans le menu **file**, choisissez l'élément **Connecting...**, remplissez les champs avec les données de connexion vers un serveur FTP que vous connaissez, puis validez en appuyant sur le bouton **Ok**. Vous devriez obtenir un écran ressemblant à celui ci-dessous.

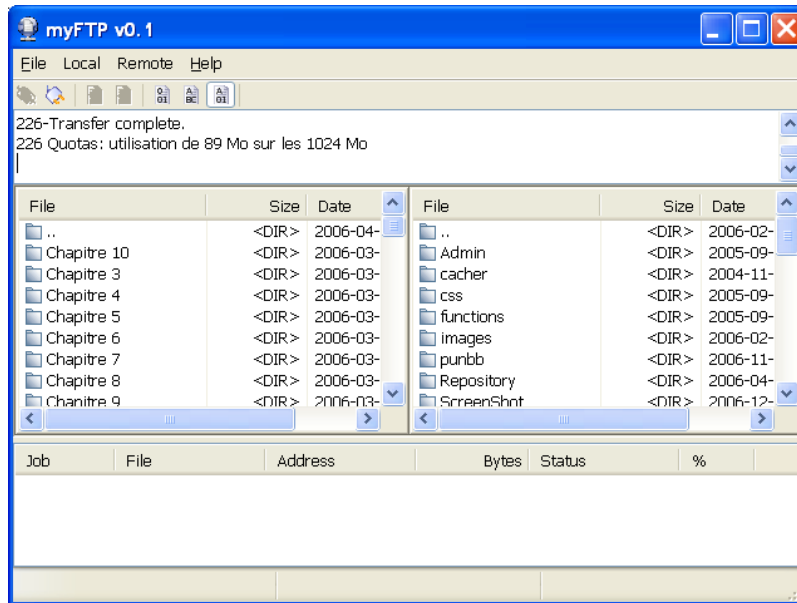


Illustration 23: wxManageRemoteThread en action

Intégration des tâches de transfert

Pour effectuer les transferts nous allons créer quatre nouvelles fonctions :

- PrepareTransfert : Récupère les données dans la liste adéquate afin de créer un élément dans la liste des transferts.
- Upload : Ajoute un élément de type « Upload » dans le liste des transferts.
- Download : Ajoute un élément de type « Download » dans la liste des transferts.
- IsAscii : Définit le mode de transfert en fonction du type de fichier.

Ouvrez le fichier *myftpframe.h* et ajoutez dans la partie publique de la classe *wxMyFTPFrame* les lignes suivantes :

```
void Upload(const wxString& path, const wxString& name);  
void Download(const wxString& path, const wxString& name,  
              const wxString& hostname, unsigned short service,  
              const wxString& user, const wxString& password, int mode);
```

Et dans la partie privée les lignes ci-dessous :

```
void PrepareTransfert(wxTransfertType type);  
bool IsAscii(const wxString& filename);
```

Puis à la fin du fichier *myftpframe.cpp*, ajoutez l'implémentation des quatre fonctions déclarées précédemment. Rien de bien compliqué, que nous ayons déjà abordé dans cet ouvrage, le code étant abondamment commenté, je ne pense pas que des explications supplémentaires soient nécessaires.

```
void wxMyFTPFrame::PrepareTransfert(wxTransfertType type)  
{
```

```

long item = -1;
long data, fcount;
wxListCtrl* list;
wxString name, path;

if (!wxGetApp().GetConnected()) // Vérification de la connexion
    return;

fcount = 0;

if (type == ttUpload) // Transfert montant
{
    path = wxGetCwd(); // Récupération du répertoire de départ
    list = m_LocalFiles; // Liste d'origine concernée par le transfert
}
else // Transfert descendant
{
    path = wxGetApp().GetRemotePath(); // Récupération du répertoire de départ
    list = m_RemoteFiles; // Liste d'origine concernée par le transfert
}

item = list->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);
// Boucle tant d'un élément sélectionné est trouvé
while (item != -1)
{
    data = list->GetItemData(item); // Récupération de la donnée associée
    if (data > 0) // donnée positive c'est donc un fichier
    {
        name = list->GetItemText(item);

        if (type == ttUpload)
            Upload(path, name);
        else
            Download(path, name, m_HostName, m_Service, m_User,
                    m_Password, m_TransfertMode);
        ++fcount;
    }
    item = list->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);
}

if (fcount)
{ // Si des éléments ont été ajoutés, un message est envoyé pour lancer le transfert
    wxCommandEvent event(wxEVT_COMMAND_MENU_SELECTED, ID_CMD_LAUNCHTHREAD);
    wxPostEvent(this, event);
}
}

void wxMyFTPFrame::Upload(const wxString& path, const wxString& name)
{
    long lCount;
    bool AsciiMode;
    wxTransJob* transjob;
    wxString remotePath;

    if (!wxGetApp().GetConnected())
        return;
    // Récupère le chemin de destination
    remotePath = wxGetApp().GetRemotePath();

    if (m_TransfertMode == TM_ASCII)
        AsciiMode = true; // Mode de transfert ASCII
    else if (m_TransfertMode == TM_BINARY)
        AsciiMode = false; // Mode transfert Binaire
    else // Sinon établi le bon mode en fonction du type de fichier (extention)
        AsciiMode = IsAscii(name);
    // création d'une instance de wxTransJob
    transjob = new wxTransJob(remotePath, path, name, ttUpload,
                            m_HostName, m_Service, m_User, m_Password,
                            AsciiMode, m_Passive);
    // Ajout d'un élément dans la liste

```

```
lCount = m_TransfertList->GetItemCount();
lCount = m_TransfertList->InsertItem(lCount, _("Upload"));
m_TransfertList->SetItem(lCount, 1, name);
m_TransfertList->SetItem(lCount, 2, m_HostName);
m_TransfertList->SetItem(lCount, 3, _T("0"));
m_TransfertList->SetItem(lCount, 4, _("Waiting"));
m_TransfertList->SetItem(lCount, 5, _T("0%"));
// Association d'une instance de wxTransJob à l'élément
m_TransfertList->SetItemData(lCount, (long)transjob);

}

void wxMyFTPFrame::Download(const wxString& path, const wxString& name,
                           const wxString& hostname, unsigned short service,
                           const wxString& user, const wxString& password,
                           int mode)
{
    bool AsciiMode;
    long lCount;
    wxTransJob* transjob;
    wxString localPath;

    localPath = wxGetCwd(); // Récupération du chemin de destination

    if (mode == TM_ASCII)
        AsciiMode = true;           // Mode de transfert ASCII
    else if (mode == TM_BINARY)
        AsciiMode = false;          // Mode de transfert Binaire
    else // Sinon établit le mode de transfert en fonction du type de fichier
        AsciiMode = IsAscii(name);
    // Création d'une instance de wxTransJob
    transjob = new wxTransJob(path, localPath, name, ttDownload, hostname,
                             service, user, password, AsciiMode, m_Passive);
    // Insertion d'un élément dans la liste
    lCount = m_TransfertList->GetItemCount();
    lCount = m_TransfertList->InsertItem(lCount, _("Download"));
    m_TransfertList->SetItem(lCount, 1, name);
    m_TransfertList->SetItem(lCount, 2, hostname);
    m_TransfertList->SetItem(lCount, 3, _T("0"));
    m_TransfertList->SetItem(lCount, 4, _("Waiting"));
    m_TransfertList->SetItem(lCount, 5, _T("0%"));
    // Association de l'instance de wxTransJob à l'élément
    m_TransfertList->SetItemData(lCount, (long)transjob);

}

bool wxMyFTPFrame::IsAscii(const wxString& filename)
{
    wxString FileExt, Exts;

    Exts = ".TXT.HTM.HTML.XML.CSS.PHP.H.HPP.C.CPP.CC.";
    FileExt = _T(".") + wxFileName(filename).GetExt().Upper() + _T(".");

    return (Exts.Find(FileExt) != -1);
}
```

Nous allons maintenant terminer l'implémentation des gestionnaires d'événements *OnLocalFilesItemActivated* et *OnRemoteFilesItemActivated* qui sont exécutés suite à une double-clic sur un élément de liste de fichiers, puis *OnMnuUploadClick* et *OnMnuDownloadClick*. Pour le premier il suffit d'ajouter la demande de transfert montant si l'élément est un fichier. Pour le second, après récupération du nom de l'élément et de la donnée associée, la vérification du signe de la donnée nous permet d'orienter soit vers un changement de répertoire distant soit d'une demande de transfert descendant. Pour le troisième et le quatrième, un simple appel de la fonction membre

PrepareTransfert avec le paramètre correspondant au sens du transfert.

```
void wxMyFTPFrame::OnLocalFilesItemActivated(wxListEvent& event)
{
    wxListItem item = event.GetItem();
    wxString name = item.GetText();
    long l = item.GetData();

    if (l < 0)
    { // C'est un répertoire, changer le répertoire courant
        wxSetWorkingDirectory(name);
        UpdateDisplay();
    }
    else // C'est un fichier demande de transfert montant
        PrepareTransfert(ttUpload);
    event.Skip();
}

void wxMyFTPFrame::OnRemoteFilesItemActivated(wxListEvent& event)
{
    wxListItem item = event.GetItem();
    wxString name = item.GetText(); // Récupération du nom
    long l = item.GetData();        // Récupération de la donnée associée

    if (l < 0) // En fonction du signe de la donnée
        SetRemoteAction(mfjChangeDir, name); // Demande de Changement du rép. distant
    else
        PrepareTransfert(ttDownload); // Demande de transfert descendant
    event.Skip();
};

void wxMyFTPFrame::OnMnuUploadClick(wxCommandEvent& event)
{
    PrepareTransfert(ttUpload);
    event.Skip();
}

void wxMyFTPFrame::OnMnuDownloadClick(wxCommandEvent& event)
{
    PrepareTransfert(ttDownload);
    event.Skip();
}
```

Pour finir ce chapitre sur les tâches, nous allons ajouter à notre classe de fenêtre principale un gestionnaire d'événements pour lancer les tâches de transfert.

Ajoutez dans la partie protégée de la déclaration de la classe *wxMyFTPFrame* la ligne ci-dessous afin de définir notre nouveau gestionnaire d'événements.

```
void OnEvtLaunchThread(wxCommandEvent& event);
```

Dans le fichier *myftpframe.cpp*, ajoutez les inclusions de fichiers suivants afin de pouvoir utiliser les classes de transfert.

```
#include "downloadthread.h"
#include "uploadthread.h"
```

Puis dans la table des gestionnaires d'événements, ajoutez le gestionnaire déclaré ci-dessus.

```
EVT_MENU(ID_CMD_LAUNCHTHREAD, wxMyFTPFrame::OnEvtLaunchThread)
```

Et enfin saisissez l'implémentation de ce nouveau gestionnaire d'événements. Celui-ci recherche un élément dans la liste qui n'a pas de tâche associée et dont l'instance de *wxMyFTP* n'est pas déjà occupée par un transfert. Si un élément est éligible à l'exécution, une instance de tâche de transfert est créée et exécutée. Si la création de la tâche échoue, l'instance de *wxMyFTP* est supprimée si nécessaire.

```
void wxMyFTPFrame::OnEvtLaunchThread(wxCommandEvent& event)
{
    long lCount, i;
    wxTransJob* TransJob;
    wxMyFTP* MyFTP = 0;
    wxTransThread* TransThread;

    lCount = m_TransfertList->GetItemCount();
    for (i = 0; i < lCount; i++) // Boucle sur les demande de transfert
    { // Récupération de la donnée associée (wxTransJob*)
        TransJob = (wxTransJob*)m_TransfertList->GetItemData(i);

        if ((TransJob != 0) && (TransJob->m_Thread == 0))
        { // Si aucune instance de tâche existe avec cet élément
            // Récupérer ou créer une instance de wxMyFTP
            MyFTP = wxGetApp().CreateFTP(TransJob->m_Hostname, TransJob->m_Service,
                                         TransJob->m_User, TransJob->m_Password,
                                         TransJob->m_Passive);

            if (!MyFTP->OnStreaming())
                break; // Aucun transfert en cours pour cet instance
            else
                MyFTP = 0; // Un transfert est déjà en cours on passe à l'élément suivant
        }
    }
    if (TransJob && MyFTP) // Tout est prêt pour établir le transfert
    { // Création d'une instance de tâche en fonction du type de transfert
        if (TransJob->m_Type == ttUpload) // Transfert montant
            TransThread = new wxUploadThread(this, MyFTP,
                                              TransJob->m_TransfertMode,
                                              TransJob->m_LocalPath,
                                              TransJob->m_RemotePath,
                                              TransJob->m_Filename);
        else // Transfert descendant
            TransThread = new wxDownloadThread(this, MyFTP,
                                              TransJob->m_TransfertMode,
                                              TransJob->m_LocalPath,
                                              TransJob->m_RemotePath,
                                              TransJob->m_Filename);

        // On lance le thread de transfert
        if (TransThread)
        { // Création de la tâche avec 4K de mémoire
            if (TransThread->Create(4096) != wxTHREAD_NO_ERROR)
            {
                m_TextCtrlInfo->AppendText(_("Can't create thread!\n"));
                TransThread->Delete();
            }
            else
            { // Association de la tâche dans les données de l'élément de liste
                TransJob->m_Thread = TransThread;
                // Ajout de la tâche dans le tableau des tâches
                wxCriticalSectionLocker enter(wxGetApp().m_array_thread_critsect);
                wxGetApp().m_threads.Add(TransThread);
                // Démarrage de la tâche
                if (TransThread->Run() != wxTHREAD_NO_ERROR)
                {
                    TransJob->m_Thread = 0;
                    m_TextCtrlInfo->AppendText(_("Can't start thread!\n"));
                    TransThread->Delete();
                }
            }
        }
    }
    else

```

```

{ // La création de l'instance de tâche à échoué
  // élimination de l'instance de wxMyFTP si nécessaire
  MyFTP->m_ftp_critsect.Enter();
  if (!MyFTP->OnStreaming() && !wxGetApp().Used(MyFTP))
  {
    wxCriticalSectionLocker enter(wxGetApp().m_array_ftp_critsect);
    wxGetApp().m_ftps.Remove(MyFTP);
    MyFTP->m_ftp_critsect.Leave();
    delete MyFTP;
  }
  else
    MyFTP->m_ftp_critsect.Leave();
}
}
event.Skip();
}

```

Vous pouvez maintenant reconstruire votre application et tester le transfert de fichier.

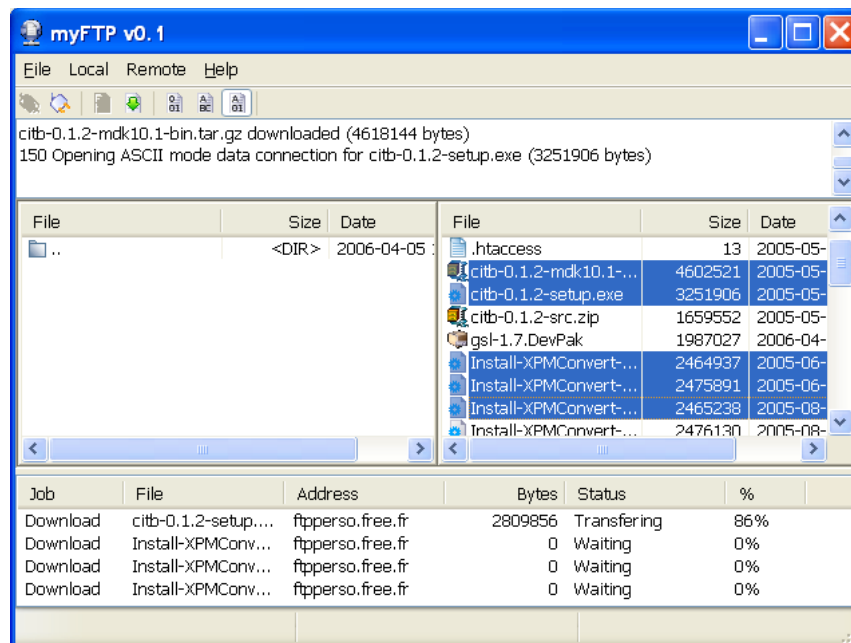


Illustration 24: Fichier en cours de transfert

! Les fichiers sources de ce chapitre sont contenus dans l'archive *Chapitre_10.zip*.

11

Le glisser/déposer (Drag and drop)

Qu'est ce que le glisser/déposer?

C'est le fait de cliquer sur un élément graphique de l'interface avec un bouton de la souris et de déplacer cet élément tout en maintenant le bouton de la souris appuyé, puis de déposer cet élément en relâchant le bouton de la souris. L'explorateur de fichier de Windows utilise cette technique pour déplacer ou copier les fichiers d'un endroit à un autre.

Les classes pour la gestion du glisser/déposer

La bibliothèque wxWidget met à notre disposition plusieurs classes pour la gestion du glisser/déposer. Celles-ci sont au nombre de onze.

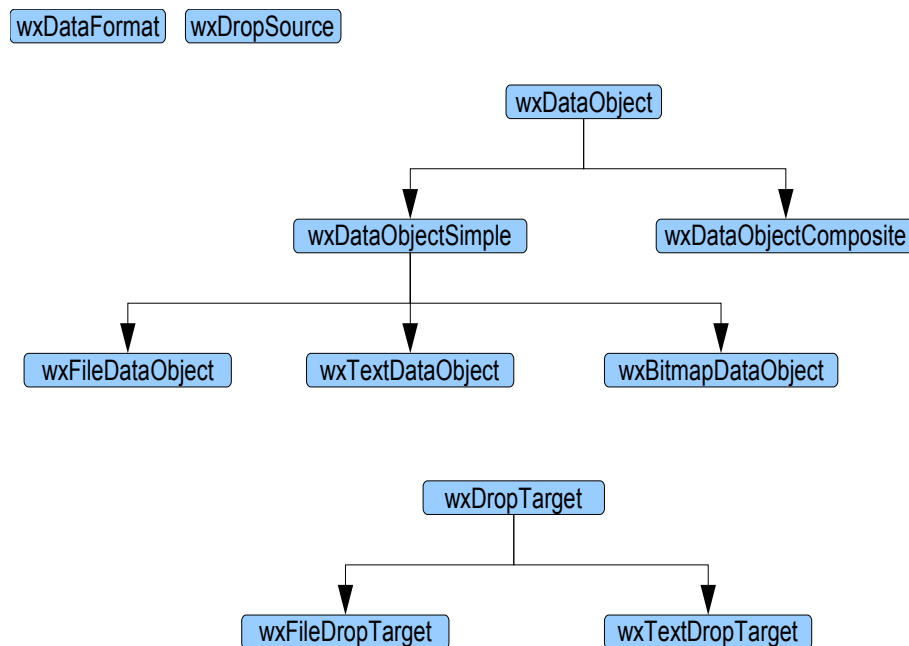


Illustration 25: Hiérarchie des classes de gestion du glisser/déposer

- `wxDataFormat` : Définition du type de donnée à transférer.
- `wxDropSource` : Représente la source d'une opération de glisser/déposer.
- `wxDataObject` : Représentation des données à transférer.
- `wxDataObjectSimple` : Données à transférer (mono-format).
- `wxDataObjectComposite` : Données à transférer (multi-format).
- `wxFileDataObject` : Classe de donnée spécialisée pour les noms de fichiers.
- `wxTextDataObject` : Classe de donnée spécialisée pour le texte.
- `wxBitmapDataObject` : Classe de donnée spécialisée pour les bitmaps.
- `wxDropTarget` : Représente une cible pour les opérations de glisser/déposer.
- `wxFileDropTarget` : Cible spécialisée pour les noms de fichiers.
- `wxTextDropTarget` : Cible spécialisée pour le texte.

Pour l'application que nous sommes entrain de développer, nous n'utiliserons que deux d'entre elles qui vont être décrites plus précisément dans les chapitres suivants.

La classe `wxDataObjectSimple`

C'est la solution la plus simple pour dériver une classe de `wxDataObject`, cet objet ne permet qu'un seul et unique format de donnée. De ce fait le nombre de fonctions virtuelles à mettre en oeuvre et réduit à trois.

- `GetDataSize` : Renvoie la taille des données.
- `GetDataHere` : Copie les données dans le tampon.
- `SetData` : Copie les données depuis le tampon.

```
virtual size_t GetDataSize()const  
virtual bool GetDataHere(void *buf)const  
virtual bool SetData(size_t len, const void *buf)
```


La classe wxDropTarget

Cette classe représente la cible dans une opération de glisser/déposer. Un objet dérivé de *wxDataObject* est associé à cette cible. Cet objet sera rempli des données de la source de l'opération de glisser/déposer, si les formats de données acceptés par l'objet *wxDataObject* correspondent au format de données de la source de l'opération de glisser/déposer. Six fonctions virtuelles peuvent être mise en oeuvre, mais au minimum cela doit être fait pour *OnData*..

- **GetData** : Copie les données de la source vers l'objet *wxDataObject* associé.
- **OnData** : Appelée après que *OnDrop* est retourné vrai.
- **OnDrop** : Appelée quand l'utilisateur relache l'objet sur une cible.
- **OnEnter** : Appelée quand la souris entre dans une cible.
- **OnDragOver** : Appelée quand on déplace la souris sur la cible.
- **OnLeave** : Appelée quand la souris quitte la cible.

```
virtual void GetData()
virtual wxDragResult OnData(wxCoord x, wxCoord y, wxDragResult def)
virtual bool OnDrop(wxCoord x, wxCoord y)
virtual wxDragResult OnEnter(wxCoord x, wxCoord y, wxDragResult def)
virtual wxDragResult OnDragOver(wxCoord x, wxCoord y, wxDragResult def)
virtual void OnLeave()
```

Création de classe pour le glisser/déposer

Toutes le classes des opérations de glisser/déposer de l'application seront déclarées dans le fichier *myftpdnd.h* et implémentées dans le fichier *myftpdnd.cpp* que vous ajouterez au projet.

Commencez par saisir les lignes ci-dessous dans le fichier *myftpdnd.h*.

```
#ifndef _DND_H_
#define _DND_H_

#if defined(__GNUG__) && !defined(__APPLE__)
#pragma interface "myftpdnd.cpp"
#endif

#include <wx/dnd.h>
#include <wx/listctrl.h>
class wxMyFTPFrame;
// Vous saisissez ici les classes de gestion du glisser/déplacer
#endif // _DND_H_
```

Puis dans le fichier *myftpdnd.cpp* :

```
#if defined(__GNUG__) && !defined(__APPLE__)
```

```
#pragma implementation "myftpdnd.h"
#endif

#include <wx/wxprec.h>

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include <wx/wx.h>
#endif

#include "myftpdnd.h"
#include "myftpframe.h"
```

La classe wxDndUploadFile

C'est la classe de donnée qui est fournie à l'objet dérivé de *wxDataObject* dans le cas d'une opération de glisser/déplacer d'une liste locale vers une liste de fichiers distants. Cet objet contient une chaîne nommée *m_Path* pour recevoir le chemin local et un tableau de chaîne nommé *m_Files* pour les noms de fichiers.

La déclaration de cette classe est la suivante :

```
class wxDndUploadFile
{
public:

    wxDndUploadFile() {}
    wxDndUploadFile(const wxString& path) : m_Path(path) {}
    wxDndUploadFile(const wxDndUploadFile& dndtransfile);
    ~wxDndUploadFile() {m_Files.Clear();}

    void AddFile(const wxString& file)
        {if (!file.IsEmpty()) m_Files.Add(file);}

    wxString& Item(size_t nIndex) const {return m_Files.Item(nIndex);}
    size_t GetCount() const {return m_Files.GetCount();}

    wxString m_Path;
    wxArrayString m_Files;
};
```

Dans le fichier *myftpdnd.cpp* vous saisissez l'implémentation du constructeur de copie.

```
wxDndUploadFile::wxDndUploadFile(const wxDndUploadFile& dndtransfile)
{
    m_Path = dndtransfile.m_Path;
    m_Files = dndtransfile.m_Files;
}
```

La classe wxDndUploadFileDataObject

Cette classe est dérivée de *wxDataObjectSimple*, elle permet de manipuler l'objet de donnée *wxDndUploadFile*, en le lisant depuis un tampon (*SetData*), en l'écrivant dans un tampon (*GetDataHere*) et en retournant la taille nécessaire du tampon

(*GetDataSize*).

Saisissez la déclaration de cette classe dans le fichier *myftpdnd.h*. Vous remarquerez que conformément au chapitre sur la classe *wxDataObjectSimple* seules les trois fonctions virtuelles citées ont été surchargées.

```
class wxDndUploadFileDataObject : public wxDataObjectSimple
{
public:

    wxDndUploadFileDataObject(wxDndUploadFile* dndtransfile = 0);
    virtual ~wxDndUploadFileDataObject() {delete m_DndTransFile;}

    virtual size_t GetDataSize() const;
    virtual bool GetDataHere(void* buf) const;
    virtual bool SetData(size_t len, const void* buf);

    wxDndUploadFile* GetTransFile() {return m_DndTransFile;}

private:

    wxDndUploadFile* m_DndTransFile;
};
```

Maintenant voici l'implémentation. Le format dans le tampon est très simple, les données chemin et noms de fichiers sont séparés par un caractère fin de ligne « \n ».

```
static const wxChar* UploadFormatId = _T("wxDndUploadFileDataObject");

wxDndUploadFileDataObject::wxDndUploadFileDataObject(wxDndUploadFile* dndtransfile)
{
    if (dndtransfile != 0) // Si aucun objet donnée n'est fourni en créer un
        m_DndTransFile = new wxDndUploadFile(*dndtransfile);
    else
        m_DndTransFile = 0;

    wxDataFormat transfileformat; // Positionner l'identifiant du format
    transfileformat.SetId(UploadFormatId);
    SetFormat(transfileformat);
}

size_t wxDndUploadFileDataObject::GetDataSize() const
{
    size_t ret, count;

    if (m_DndTransFile) // Vérification de l'existence d'un objet donnée
    { // Chemin et noms de fichiers séparés par \n
        ret = m_DndTransFile->m_Path.Len() + 1; // On ajoute 1 caractère pour \n
        count = m_DndTransFile->GetCount();
        ret += count; // On ajoute count caractère \n
        for (size_t i = 0; i < count; i++)
            ret += m_DndTransFile->Item(i).Len();
    }
    else
        ret = 0;
    return ret;
}

bool wxDndUploadFileDataObject::GetDataHere(void* buf) const
{
    if (m_DndTransFile) // Vérification de l'existence d'un objet donnée
    {
        wxString tmp;
        size_t count;
        // remplissage du tampon en séparant par des \n
        tmp = m_DndTransFile->m_Path + _T("\n"); // le premier est le chemin
        count = m_DndTransFile->GetCount();
```

```
        for (size_t i = 0; i < count; i++) // les autres sont les noms de fichiers
            tmp += m_DndTransFile->Item(i) + _T("\n");
        memcpy(buf, tmp.c_str(), tmp.Len());
        return true;
    }
    return false;
}

bool wxDndUploadFileDataObject::SetData(size_t len, const void* buf)
{
    if (len != 0)
    {
        wxString tmp1, tmp2;

        if (!m_DndTransFile) // Créer un objet donnée si inexistant
            m_DndTransFile = new wxDndUploadFile;
        tmp1 = wxString((const char*)buf, len);
        tmp2 = tmp1.BeforeFirst(_T('\n')); // découpage du tampon à chaque \n
        tmp1 = tmp1.AfterFirst(_T('\n'));
        m_DndTransFile->m_Path = tmp2; // Le premier est le chemin
        m_DndTransFile->m_Files.Clear();
        while (!tmp1.IsEmpty())
        {
            tmp2 = tmp1.BeforeFirst(_T('\n')); // découpage du tampon à chaque \n
            tmp1 = tmp1.AfterFirst(_T('\n'));
            if (!tmp2.IsEmpty())
                m_DndTransFile->AddFile(tmp2); // Les autres sont les noms de fichiers
        }
    }

    return true;
}
```

La classe wxDndRemoteTarget

Cette classe représente la cible du glisser/déplacer, elle est issue de la classe *wxDropTarget*. Comme vous pourrez le voir dans la déclaration de la classe seule deux fonctions virtuelles ont été surchargées, il s'agit de *OnDragOver* et *OnData*. Le constructeur de la classe stocke les données *wxListCtrl* et *wxMyFTPFrame* fournies correspondant respectivement à la liste de fichiers distant et à la fenêtre principale de l'application. Puis une instance d'objet *wxDndUploadFileDataObject* est créée et fournie au constructeur de la classe parente.

```
class wxDndRemoteTarget : public wxDropTarget
{
public:

    wxDndRemoteTarget(wxListCtrl* owner, wxMyFTPFrame* frame) :
        wxDropTarget(new wxDndUploadFileDataObject)
    {
        m_Owner = owner;
        m_Frame = frame;
    }

    virtual wxDragResult OnDragOver(wxCoord x, wxCoord y, wxDragResult def);
    virtual wxDragResult OnData(wxCoord x, wxCoord y, wxDragResult def);

private:

    wxListCtrl* m_Owner;
    wxMyFTPFrame* m_Frame;
};
```

Vous remarquerez que la surcharge de la fonction virtuelle *OnDragOver* est faite uniquement pour indiquer que l'opération de glisser/déplacer ne générera que des copies et pas de déplacement. Quant à la fonction *OnData*, elle récupère les données issues de l'opération de glisser/déplacer et injecte les noms de fichiers avec le chemin dans la fonction *Upload* membre *wxMyFTPFrame*.

```
wxDragResult wxDndRemoteTarget::OnDragOver(wxCoord x, wxCoord y, wxDragResult def)
{
    if (!wxGetApp().GetConnected()) // Si aucune connection active
        return wxDragNone;         // on ne peut pas déposer
    // Pas de déplacements uniquement des copies
    return (def == wxDragMove ? wxDragCopy : def);
}

wxDragResult wxDndRemoteTarget::OnData(wxCoord x, wxCoord y, wxDragResult def)
{
    if (!GetData()) // récupération des données
        return wxDragNone;

    wxDndUploadFile* TransFile;
    // Récupération d'un pointeur sur l'objet données (wxDndUploadFile)
    TransFile = ((wxDndUploadFileDataObject *)GetDataObject())->GetTransFile();
    size_t nFiles = TransFile->GetCount();
    // Appel de la fonction Upload de la fenêtre principale pour
    // chaque nom de fichier dans l'instance d'objet wxDndUploadFile
    for (size_t i = 0; i < nFiles; i++)
        m_Frame->Upload(TransFile->m_Path, TransFile->Item(i));

    if (nFiles)
    { // Si des fichiers étaient présents, envoie d'un message pour lancer la tâche
        wxCommandEvent event(wxEVT_COMMAND_MENU_SELECTED, ID_CMD_LAUNCHTHREAD);
        wxPostEvent(m_Frame, event);
    }

    return wxDragCopy;
}
```

La classe wxDndDownloadFile

C'est la classe de donnée qui est transmise à l'instance de la classe *wxDndDownloadFileDataObject* lors d'une opération de glisser/déposer d'une liste de fichiers distants vers une liste de fichiers locaux. Cette classe contient plus de donnée que la classe *wxDndUploadFile* car il lui faut conserver les paramètres de connexion au serveur ftp en plus du chemin et des noms de fichiers transmis. Néanmoins la déclaration de cette classe est sensiblement similaire à celle de la classe *wxDndUploadFile*.

```
class wxDndDownloadFile
{
public:

    wxDndDownloadFile() : m_Service(0) {}
    wxDndDownloadFile(const wxString& path, const wxString& hostname,
                      unsigned short service, const wxString& user,
                      const wxString& password, int mode) :
        m_Path(path), m_Hostname(hostname), m_Service(service), m_User(user),
        m_Password(password), m_Mode(mode) {}

    wxDndDownloadFile(const wxDndDownloadFile& dndtransfile);
```

```
~wxDndDownloadFile() {m_Files.Clear();}

void AddFile(const wxString& file)
    {if (!file.IsEmpty()) m_Files.Add(file);}

wxString& Item(size_t nIndex) const {return m_Files.Item(nIndex);}
size_t GetCount() const {return m_Files.GetCount();}

wxString m_Path;
wxString m_Hostname;
unsigned short m_Service;
wxString m_User;
wxString m_Password;
int m_Mode;
wxArrayString m_Files;
};
```

L'implémentation de cette classe se limite à celle du constructeur de copie.

```
wxDndDownloadFile::wxDndDownloadFile(const wxDndDownloadFile& dndtransfile)
{
    m_Path = dndtransfile.m_Path;
    m_Hostname = dndtransfile.m_Hostname;
    m_Service = dndtransfile.m_Service;
    m_User = dndtransfile.m_User;
    m_Password = dndtransfile.m_Password;
    m_Mode = dndtransfile.m_Mode;
    m_Files = dndtransfile.m_Files;
}
```

La classe wxDndDownloadFileDataObject

Comme pour la classe *wxDndUploadFileDataObject* trois fonctions virtuelles sont surchargées. La seule différence est que celles-ci gèrent plus de données lors du transfert depuis ou vers le tampon.

```
class wxDndDownloadFileDataObject : public wxDataObjectSimple
{
public:

    wxDndDownloadFileDataObject(wxDndDownloadFile* dndtransfile = 0);
    virtual ~wxDndDownloadFileDataObject() {delete m_DndTransFile;}

    virtual size_t GetDataSize() const;
    virtual bool GetDataHere(void* buf) const;
    virtual bool SetData(size_t len, const void* buf);

    wxDndDownloadFile* GetTransFile() {return m_DndTransFile;}

private:

    wxDndDownloadFile* m_DndTransFile;
};
```

La petite différence que l'on peut constater dans l'implémentation des fonctions de transfert ou de calcul de taille du tampon, est que la première donnée n'est pas suivie d'un caractère nouvelle ligne « \n ». C'est parce que la taille de cette donnée ne fait qu'un seul et unique caractère.

```
static const wxString* DownloadFormatId = _T("wxDndDownloadFileDataObject");
```

```
wxDndDownloadFileDataObject::wxDndDownloadFileDataObject(wxDndDownloadFile* dndtransfile)
{
    if (dndtransfile != 0) // Si aucun objet donnée n'est fourni en créer un
        m_DndTransFile = new wxDndDownloadFile(*dndtransfile);
    else
        m_DndTransFile = 0;

    wxDataFormat transfileformat; // Positionner l'identifiant du format
    transfileformat.SetId(DownloadFormatId);
    SetFormat(transfileformat);
}

size_t wxDndDownloadFileDataObject::GetDataSize() const
{
    size_t ret, count;
    wxString tmp;

    if (m_DndTransFile)
    {
        ret = 1; // pour m_Mode (Binary, ascIi, Auto)
        tmp = wxString::Format(_T("%u"), (unsigned int)m_DndTransFile->m_Service);
        ret += tmp.Len() + 1; // On ajoute 1 caractère pour \n
        ret += m_DndTransFile->m_Path.Len() + 1; // On ajoute 1 caractère pour \n
        ret += m_DndTransFile->m_Hostname.Len() + 1; // On ajoute 1 caractère pour \n
        ret += m_DndTransFile->m_User.Len() + 1; // On ajoute 1 caractère pour \n
        ret += m_DndTransFile->m_Password.Len() + 1; // On ajoute 1 caractère pour \n

        count = m_DndTransFile->GetCount();
        ret += count; // On ajoute count caractère \n
        for (size_t i = 0; i < count; i++)
            ret += m_DndTransFile->Item(i).Len();
    }
    else
        ret = 0;
    return ret;
}

bool wxDndDownloadFileDataObject::GetDataHere(void* buf) const
{
    if (m_DndTransFile)
    {
        wxString tmp;
        size_t count;
        // Un caractère pour le mode de transfert
        switch (m_DndTransFile->m_Mode)
        {
            case TM_ASCII : tmp = _T("I");
                break;
            case TM_BINARY : tmp = _T("B");
                break;
            case TM_AUTO : tmp = _T("A");
                break;
            default : return false;
        }
        // Puis le reste des éléments sont séparés par \n
        tmp += wxString::Format(_T("%u\n"), (unsigned int)m_DndTransFile->m_Service);
        tmp += m_DndTransFile->m_Path + _T("\n");
        tmp += m_DndTransFile->m_Hostname + _T("\n");
        tmp += m_DndTransFile->m_User + _T("\n");
        tmp += m_DndTransFile->m_Password + _T("\n");
        // ON ajoute les noms de fichiers séparés par \n
        count = m_DndTransFile->GetCount();
        for (size_t i = 0; i < count; i++)
            tmp += m_DndTransFile->Item(i) + _T("\n");
        memcpy(buf, tmp.c_str(), tmp.Len());
        return true;
    }
    return false;
}
```

```
bool wxDndDownloadFileDataObject::SetData(size_t len, const void* buf)
{
    if (len != 0)
    {
        wxString tmp1, tmp2, srv;
        unsigned long usrv;

        if (!m_DndTransFile)
            m_DndTransFile = new wxDndDownloadFile;
        tmp1 = wxString((const char*)buf, len);

        switch (tmp1.GetChar(0))
        {
            case _T('I') : m_DndTransFile->m_Mode = TM_ASCII;
                break;
            case _T('B') : m_DndTransFile->m_Mode = TM_BINARY;
                break;
            case _T('A') : m_DndTransFile->m_Mode = TM_AUTO;
                break;
            default : return false;
        }
        tmp2 = tmp1.BeforeFirst(_T('\n')).Mid(1);
        tmp1 = tmp1.AfterFirst(_T('\n'));
        srv = tmp2;
        if (!srv.ToULong(&usrv))
            return false;
        m_DndTransFile->m_Service = (unsigned short)usrv;
        tmp2 = tmp1.BeforeFirst(_T('\n'));
        tmp1 = tmp1.AfterFirst(_T('\n'));
        m_DndTransFile->m_Path = tmp2;
        tmp2 = tmp1.BeforeFirst(_T('\n'));
        tmp1 = tmp1.AfterFirst(_T('\n'));
        m_DndTransFile->m_Hostname = tmp2;
        tmp2 = tmp1.BeforeFirst(_T('\n'));
        tmp1 = tmp1.AfterFirst(_T('\n'));
        m_DndTransFile->m_User = tmp2;
        tmp2 = tmp1.BeforeFirst(_T('\n'));
        tmp1 = tmp1.AfterFirst(_T('\n'));
        m_DndTransFile->m_Password = tmp2;

        m_DndTransFile->m_Files.Clear();
        while (!tmp1.IsEmpty())
        {
            tmp2 = tmp1.BeforeFirst(_T('\n'));
            tmp1 = tmp1.AfterFirst(_T('\n'));
            if (!tmp2.IsEmpty())
                m_DndTransFile->AddFile(tmp2);
        }
        return true;
    }
    return false;
}
```

La classe wxDndLocaleTarget

Elle représente la cible d'une opération de glisser/déposer lorsque cette opération se fait depuis une liste de fichiers distants vers une liste de fichiers locaux. Comme pour la classe *wxDndRemoteTarget* deux même fonctions virtuelles sont surchargées.

```
class wxDndLocaleTarget : public wxDropTarget
{
public:

    wxDndLocaleTarget(wxListCtrl* owner, wxMyFTPFrame* frame) :
        wxDropTarget(new wxDndDownloadFileDataObject)
    {
```

```
m_Owner = owner;
m_Frame = frame;
}

virtual wxDragResult OnDragOver(wxCoord x, wxCoord y, wxDragResult def);
virtual wxDragResult OnData(wxCoord x, wxCoord y, wxDragResult def);

private:

wxListCtrl* m_Owner;
wxMyFTPFrame* m_Frame;
};
```

L'implémentation des deux fonctions virtuelles *OnDragOver* et *OnData* est sensiblement similaire dans leur structure à celle de la classe *wxDndRemoteTarget*.

```
wxDragResult wxDndLocaleTarget::OnDragOver(wxCoord x, wxCoord y, wxDragResult def)
{ // Pas de déplacements uniquement des copies
    return (def == wxDragMove ? wxDragCopy : def);
}

wxDragResult wxDndLocaleTarget::OnData(wxCoord x, wxCoord y, wxDragResult def)
{
    if (!GetData()) // récupération des données
        return wxDragNone;

    wxDndDownloadFile* TransFile;
    // Récupération d'un pointeur sur l'objet données (wxDndDownloadFile)
    TransFile = ((wxDndDownloadFileDataObject *)GetDataObject())->GetTransFile();
    // Appel de la fonction Download de la fenêtre principale pour
    // chaque nom de fichier dans l'instance d'objet wxDndDownloadFile
    size_t nFiles = TransFile->GetCount();
    for (size_t i = 0; i < nFiles; i++)
        m_Frame->Download(TransFile->m_Path, TransFile->Item(i),
                          TransFile->m_Hostname, TransFile->m_Service,
                          TransFile->m_User, TransFile->m_Password,
                          TransFile->m_Mode);

    if (nFiles)
    { // Si des fichiers était présent, envoi d'un message pour lancer la tâche
        wxCommandEvent event(wxEVT_COMMAND_MENU_SELECTED, ID_CMD_LAUNCHTHREAD);
        wxPostEvent(m_Frame, event);
    }

    return wxDragCopy;
}
```

Intégration des classes de glisser/déposer

Maintenant que les classes pour la gestion du glisser/déposer sont terminées, nous allons faire en sorte qu'elles fonctionnent parfaitement avec le reste de notre application. Pour ce faire nous commencerons par ajouter une ligne dans le fichier *myftpframe.cpp* pour l'inclusion du fichier *myftpdnd.h*.

Puis nous ajouterons à la fin de l'implémentation de la fonction *CreateControls* membre de la classe *wxMyFTPFrame* afin d'indiquer quel instance d'objet cible utiliser pour les listes de fichiers lors d'une opération de glisser/déposer.

```
// Indication de l'objet cible pour les opération de glisser/déplacer
m_LocalFiles->SetDropTarget(new wxDndLocaleTarget(m_LocalFiles, this));
m_RemoteFiles->SetDropTarget(new wxDndRemoteTarget(m_RemoteFiles, this));
```

Enfin nous compléterons les gestionnaires d'événements *OnLocalFilesBeginDrag* et *OnRemoteFilesBeginDrag* pour qu'ils déclenchent les opérations de glisser/déposer le moment venu.

```
void wxMyFTPFrame::OnLocalFilesBeginDrag(wxListEvent& event)
{
    long item = -1, data;
    wxDndUploadFile* transfile;

    transfile = new wxDndUploadFile(wxGetCwd()); // Création d'une donnée à glisser
    // remplissage de la donnée avec les noms de fichiers sélectionnés
    item = m_LocalFiles->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);
    while (item != -1)
    {
        data = m_LocalFiles->GetItemData(item);
        if (data > 0) // c'est un fichier
            transfile->AddFile(m_LocalFiles->GetItemText(item));
        item = m_LocalFiles->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);
    }
    if (transfile->GetCount() == 0) // Si aucun fichier détruire l'instance et sortir
    {
        delete transfile;
        return;
    }

    wxDndUploadFileDataObject UploadData(transfile); // Créer un objet de manipulation ↴
de données

    wxDropSource source(UploadData, this, wxDROP_ICON(dnd_copy), // Créer une source de D&D
                        wxDROP_ICON(dnd_move), wxDROP_ICON(dnd_none));

    source.DoDragDrop(wxDrag_DefaultMove); // Effectuer l'opération de D&D

    event.Skip();
}

void wxMyFTPFrame::OnRemoteFilesBeginDrag(wxListEvent& event)
{
    long item = -1, data;
    wxDndDownloadFile* transfile;
    // Créer une instance d'objet de donnée de D&D
    transfile = new wxDndDownloadFile(wxGetApp().GetRemotePath(), m_HostName,
                                    m_Service, m_User, m_Password, m_TransfertMode);
    // remplir cette donnée avec les noms de fichiers sélectionnés
    item = m_RemoteFiles->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);
    while (item != -1)
    {
        data = m_RemoteFiles->GetItemData(item);
        if (data > 0) // c'est un fichier
            transfile->AddFile(m_RemoteFiles->GetItemText(item));
        item = m_RemoteFiles->GetNextItem(item, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);
    }
    if (transfile->GetCount() == 0) // Si aucun fichier détruire l'instance et sortir
    {
        delete transfile;
        return;
    }
    // Créer un objet pour la manipulation de la donnée
    wxDndDownloadFileDataObject DownloadData(transfile);
    // Créer un objet source de l'opération de D&D
    wxDropSource source(DownloadData, this, wxDROP_ICON(dnd_copy),
                        wxDROP_ICON(dnd_move), wxDROP_ICON(dnd_none));

    source.DoDragDrop(wxDrag_DefaultMove); // Effectuer l'opération de D&D

    event.Skip();
}
```

Vous pouvez à présent reconstruire l'application et tester le glisser/déposer.

N'oublier pas que vous devez être connecté avant de pouvoir déposer des fichiers de la liste locale dans la liste distante et inversement.



Les fichiers sources de ce chapitre sont contenus dans l'archive *Chapitre_11.zip*.

12 Internationalisation

Vous trouverez souvent ce terme sous l'abréviation I18N. Pourquoi I18N? Simplement car dix-huit lettres séparent le premier I du dernier N. Nous allons voir dans ce chapitre comment internationaliser notre application au niveau des textes uniquement, il est évident que pour une internationalisation complète il faudrait aussi gérer le format des dates et heures et l'alignement pour les langues s'écrivant de droite à gauche.

Vous l'avez sans doute remarqué mais tout les textes contenus dans les sources de cet ouvrage sont encadrés par les macro `_()` ou `_T()`. La première génère du code pour les textes devant être traduits, la deuxième pour les textes ne devant pas être traduits.

Pour lire les textes traduits wxWidgets à besoin de catalogues de messages. Ce sont des fichiers binaires dont l'extension est « mo ». Pour créer ces fichiers binaires vous avez besoin de fichiers catalogues sources dont l'extension « po » et d'un utilitaire pour les transformer en catalogues binaires. Plusieurs utilitaires pour créer et transformer les fichiers catalogues sources existent. Nous allons voir l'utilisation de l'un d'entre eux dans le chapitre suivant.

Utilisation de poEdit

L'utilitaire poEdit est un logiciel libre écrit avec la bibliothèque wxWidgets qui permet d'éditer des catalogues de textes sources et de les transformer en catalogues de textes binaires.

Si vous ne l'avez pas encore fait, téléchargez et installez le logiciel poEdit. Vous trouverez l'adresse du site au début de cet ouvrage au chapitre « Utilitaires divers ». Puis lancez ce logiciel, une fenêtre similaire à celle représentée ci-dessous devrait s'afficher.

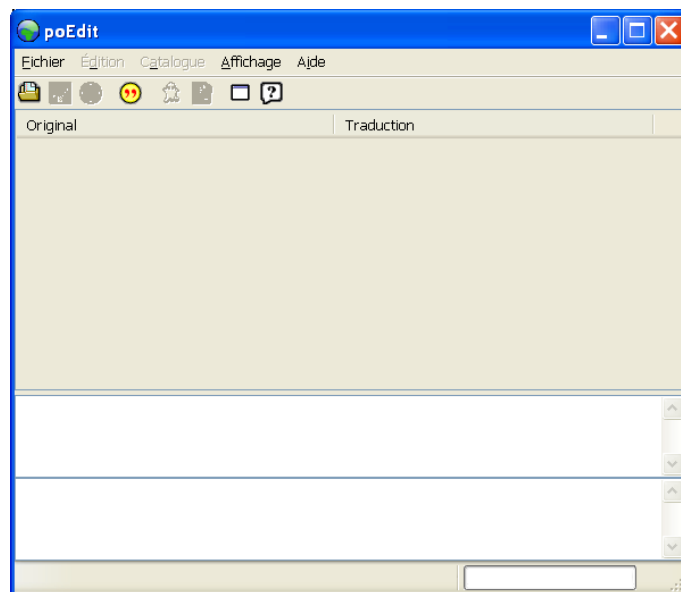


Illustration 26: Le logiciel poEdit

Sélectionnez l'élément **Nouveau catalogue** dans le menu **Fichier**. Puis remplissez les champs de la boîte de dialogue qui s'affiche, saisissez le nom et la version du projet, le jeu de caractères utilisé dans le catalogue et enfin le jeu de caractère de la source.

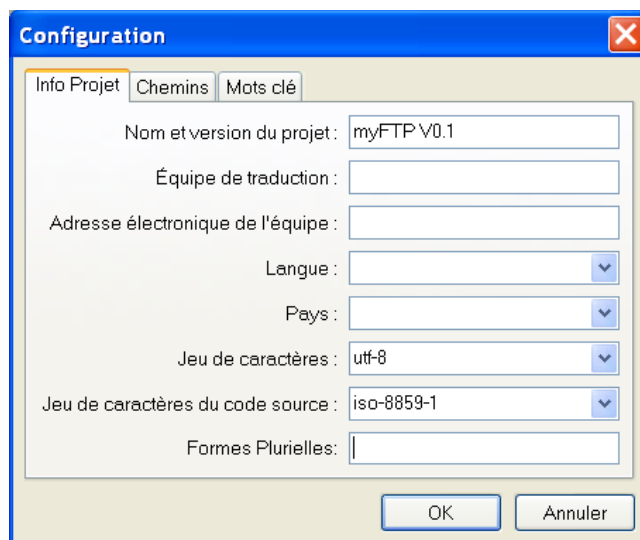


Illustration 27: poEdit configuration - Info Projet

Sélectionnez l'onglet **Chemins** et saisissez les valeurs « . » pour le chemin de base et « .. » dans la liste des chemins. Puis validez en appuyant sur le bouton **OK**.

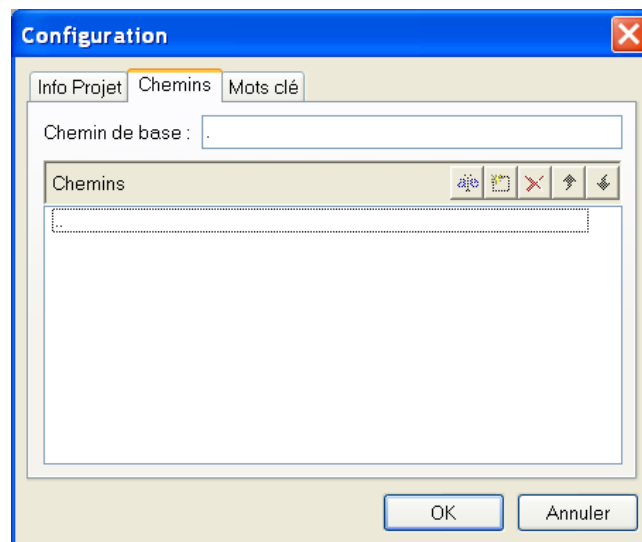


Illustration 28: poEdit configuration - Chemins

Une boîte de dialogue d'enregistrement de fichier s'ouvre, sélectionnez le chemin d'accès à votre projet qui doit être *Tutoriel wxWidgets*, puis créez un nouveau répertoire nommé *locale* et enregistrez dans ce nouvel emplacement le fichier catalogue source que vous nommerez *default.po*. Un traitement est effectué afin de récupérer toutes les chaînes de texte à traduire de votre projet, puis une boîte de dialogue s'affiche pour vous informer du résultat du traitement. Validez ce résultat en appuyant sur la touche [ENTRÉE] ou en appuyant avec le pointeur de votre souris sur le bouton **OK**.

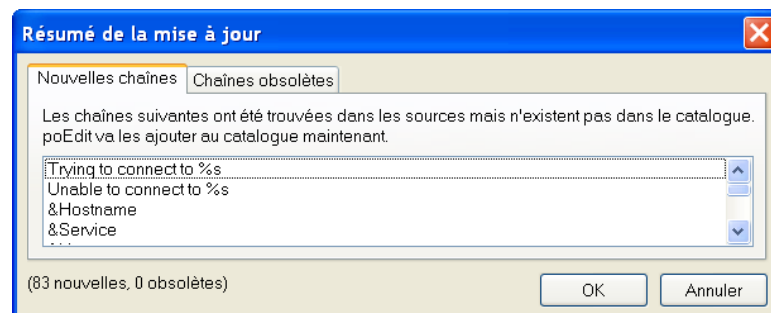


Illustration 29: poEdit - Résultat du traitement des sources

Toutes les chaînes traitées sont ajoutées dans le catalogue, la fenêtre de poEdit est maintenant remplie de textes à traduire.

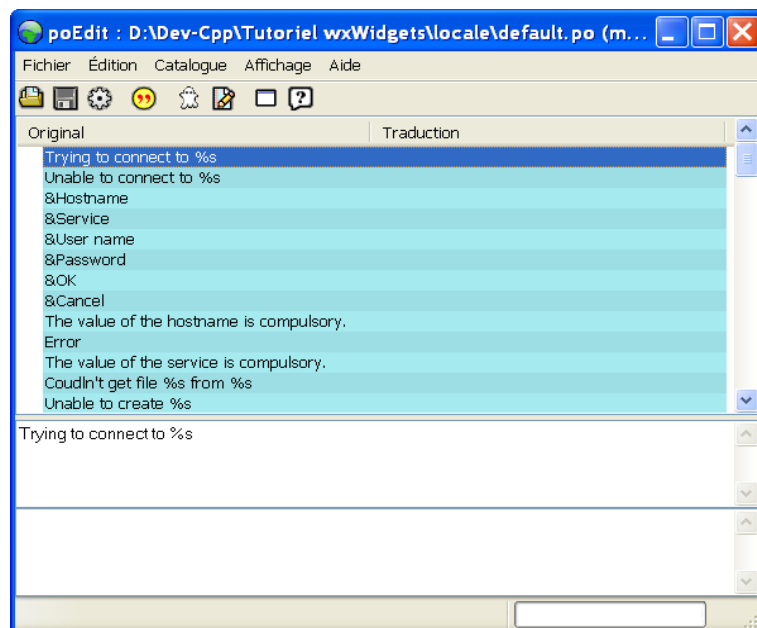


Illustration 30: poEdit - Texte à traduire

Appuyez sur les touches [CTRL]+[S] pour sauvegarder votre catalogue, puis quitter l'application. Vous avez à présent deux fichiers catalogues un binaire et un source comme l'indique la vue ci-dessous.

Nom	Taille	Type	Date de modification	Attributs
default.po	17 Ko	Gettext message catalog	12/04/2006 11:53	A
default.mo	1 Ko	Fichier MO	12/04/2006 11:53	A

Illustration 31: Fichiers catalogues

Le fichier à l'extension PO vous servira de base pour l'élaboration de catalogue de chaque langue à traduire. Vous pourrez aussi fournir ce fichier avec vos binaires afin que l'utilisateur puisse traduire la catalogue dans sa langue si cela n'est pas encore fait.

Nous allons maintenant effectuer la traduction du catalogue en français. Dans le répertoire *locale* créez un sous-répertoire *fr* puis entrez dans ce nouveau répertoire et créez un autre sous-répertoire nommé cette fois *LC_MESSAGES*. Copiez à présent le fichier *default.po* dans cet emplacement en le renommant *myftp.po*.

Nom	Taille	Type	Date de modification	Attributs
myftp.po	17 Ko	Gettext message catalog	12/04/2006 11:53	A

Illustration 32: Arborescence des répertoires des catalogues

Ouvrez ce nouveau fichier, choisissez dans le menu **Catalogue**, l'élément **Configuration**, puis dans la liste **Langue** sélectionnez **French** et dans la liste **Pays** sélectionnez **FRANCE**. Enfin validez en utilisant le bouton **OK**.

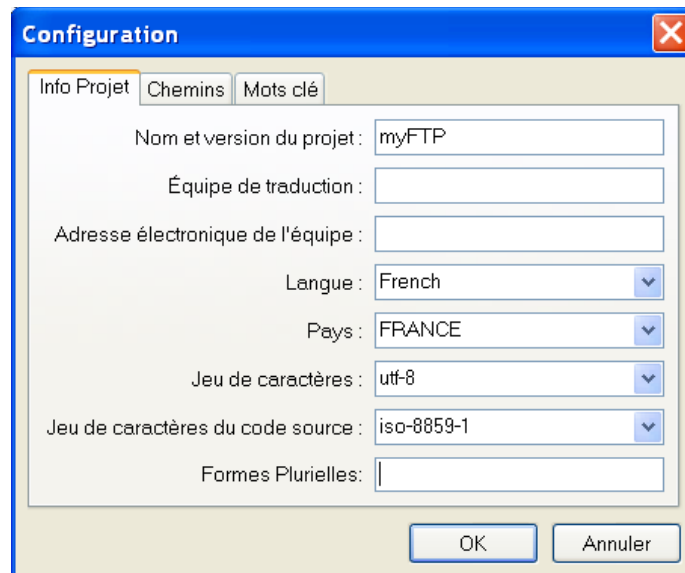


Illustration 33: poEdit - configuration langue et pays

Passons maintenant à la traduction, sélectionnez la première ligne dans la liste, puis saisissez la traduction dans l'éditeur en bas de la fenêtre. Une fois la traduction de cet élément fait, sélectionnez l'élément suivant à traduire. Si vous souhaitez copier l'élément dans l'éditeur du bas, appuyez sur les touches **[ALT]+[C]**. Pensez de temps en temps à sauvegarder en appuyant sur les touches **[CTRL]+[S]** ou en utilisant l'élément Enregistrer du menu Fichier, ou encore en cliquant sur l'icône représentant une disquette dans la barre d'outils.

Vous trouverez les traductions de la bibliothèque wxWidgets à cet adresse :
<http://www.wxwidgets.org/i18n.php>.

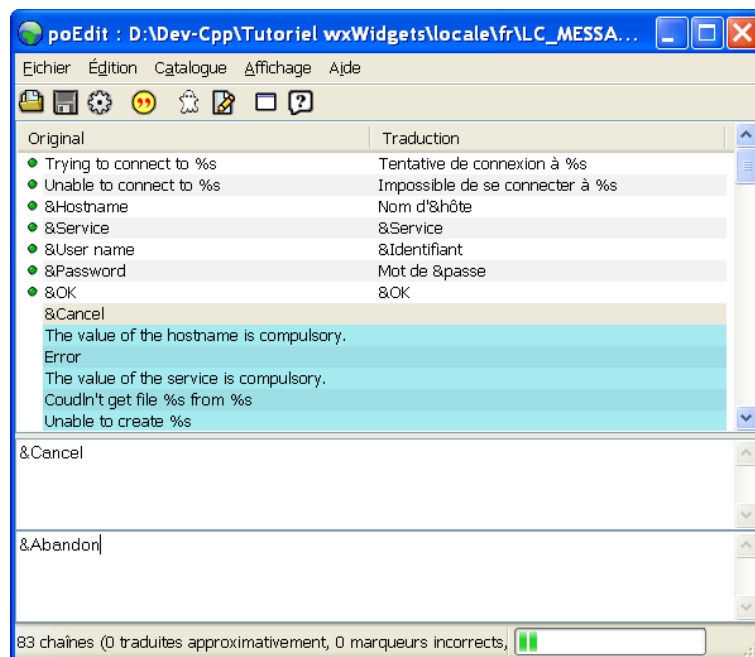


Illustration 34: poEdit - Traduction en cours

La classe wxLocale

Cette classe permet de gérer les catalogues de messages qui contiennent les traductions des chaînes à utiliser pour la langue courante. Nous n'utiliserons que trois fonctions membres de cette classe :

- AddCatalogLookupPathPrefix : Ajoute un préfixe au chemin de recherche des catalogues
- Init : Initialisation
- AddCatalog : Ajoute un catalogue de messages

Intégration de l'internationalisation

Pour commencer nous allons ajouter une instance de la classe *wxLocale* dans la partie privée de la déclaration de la classe *wxMyFTPApp*.

```
wxLocale m_locale;
```

Puis nous ajouterons dans le fichier *myftpapp.cpp* la ligne d'inclusion suivante :

```
#include <wx/intl.h>
```

Enfin dans l'implémentation de la fonction *OnInit* membre de la classe *wxMyFTPApp*, nous ajouterons juste après l'appel de la fonction *OnInit* de la classe

parente les lignes suivantes afin de définir de nouveaux préfixes de chemin, d'initialiser l'instance de classe `wxLocale` au langage du système, et d'ajouter des catalogues de messages.

```
// Gestion de la langue
// Ajout des préfixes possibles de chemins d'accès aux catalogues
wxLocale::AddCatalogLookupPathPrefix(wxT("."));
wxLocale::AddCatalogLookupPathPrefix(wxT(".."));
wxLocale::AddCatalogLookupPathPrefix(_T("locale"));
// Mise en place de la langue par défaut du système
m_locale.Init(wxLANGUAGE_DEFAULT);
{
    wxLogNull noLog; // Supprime les erreurs si les catalogues n'existent pas
    // Catalogue de l'application
    m_locale.AddCatalog(_T("myftp"));
    // Catalogue de wxWidgets
    m_locale.AddCatalog(_T("wxstd"));
}
```

Vous pouvez maintenant reconstruire votre application et vérifier que celle-ci est bien traduite en français comme le montre l'illustration ci-dessous.

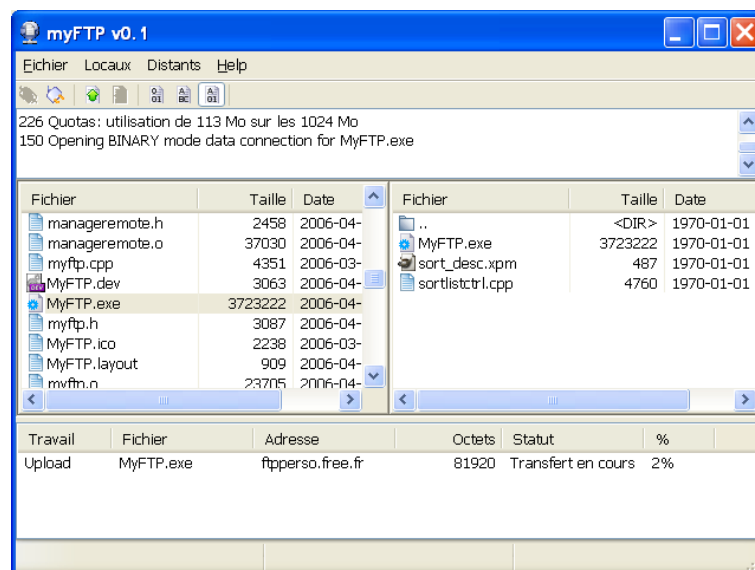


Illustration 35: Application traduite en français



Les fichiers sources de ce chapitre et les fichiers catalogues sont contenus dans l'archive *Chapitre_12.zip*.

Illustration 1: MyFTP 0.1.0 (Windows XP).....	1
Illustration 2: Mise à jour Dev-C++.....	7
Illustration 3: Dialogue de mise à jour de Dev-C++.....	7
Illustration 4: Lancement de Dev-C++.....	9
Illustration 5: Menu nouveau projet.....	9
Illustration 6: Dialogue de création d'un nouveau projet.....	10
Illustration 7: Dialogue de sauvegarde du projet.....	10
Illustration 8: Dialogue de suppression du projet.....	11
Illustration 9: Fenêtre principale de l'application.....	14
Illustration 10: Fenêtre principale de MyFTP.....	19
Illustration 11: Liste des fichiers images après désarchivage de images.zip.....	21
Illustration 12: Fichiers à supprimer.....	21
Illustration 13: Fenêtre principale de MyFTP à la fin du chapitre 5.....	25
Illustration 14: Schéma de positionnement des contrôles et des wxSizer.....	30
Illustration 15: Fenêtre de MyFTP à la fin du chapitre 6.....	33
Illustration 16: Positionnement des contrôles avec les wxSizer.....	37
Illustration 17: Le dialogue de connexion.....	41
Illustration 18: Exécution de la commande About.....	47
Illustration 19: Affichage des fichiers dans la liste locale.....	52
Illustration 20: MyFTP à la fin du chapitre 7.....	57
Illustration 21: Hiérarchie des classes dérivées de wxSocketBase.....	59
Illustration 22: Hiérarchie des classes de tâches.....	75
Illustration 23: wxManageRemoteThread en action.....	98
Illustration 24: Fichier en cours de transfert.....	103
Illustration 25: Hiérarchie des classes de gestion du glisser/déposer.....	105
Illustration 26: Le logiciel poEdit.....	120
Illustration 27: poEdit configuration - Info Projet.....	120
Illustration 28: poEdit configuration - Chemins.....	121
Illustration 29: poEdit - Résultat du traitement des sources.....	121
Illustration 30: poEdit - Texte à traduire.....	122
Illustration 31: Fichiers catalogues.....	122
Illustration 32: Arborescence des répertoires des catalogues.....	122
Illustration 33: poEdit - configuration langue et pays.....	123

Illustration 34: poEdit - Traduction en cours.....	124
Illustration 35: Application traduite en français.....	125

Je remercie celles et ceux qui m'ont supporté pendant la rédaction de ces quelques pages ainsi que ceux qui m'ont aidé à la rédaction ou à la correction. J'espère ne pas en oublier.

Yoann Sculo (Aquanum), Nicolas Boumal (Kirua), Jean-Clément Devaux (Patemino), Marie Gambini, Adrien Cailleau-Lepetit.

Document réalisé avec OpenOffice.org 2.0

<http://fr.openoffice.org/>