

# Installation des DevPacks wxWidgets 2.6.3 pour Code::Blocks

## 1) Télécharger les DevPacks à installer :

Je vous recommande d'aller sur le site de FredCL : <http://cfred.free.fr/>

Il faut télécharger le DevPack des fichiers en-têtes et au minimum un des 4 DevPacks contenant les bibliothèques compilées. Vous pouvez, si vous n'êtes pas sûr, télécharger les 4, ils ne sont pas très gros. Vous trouverez également un DevPack contenant les exemples et l'aide au format MS-Help.

## 2) Décompression du contenu des DevPacks :

Les DevPacks sont en fait des archives tar compressées au format bz2. Il vous suffit de les renommer en changeant leur extension en « **.tar.bz2** ».

Ainsi, par exemple, pour les fichiers en-tête, vous devriez obtenir un fichier qui porte le nom suivant :

« **wxWidgets\_263\_headers.tar.bz2** »

Il ne vous reste plus qu'à les décompresser avec un logiciel tel que WinRAR, ou mieux, 7-Zip qui lui est totalement gratuit :



## 3) Répertoire d'installation

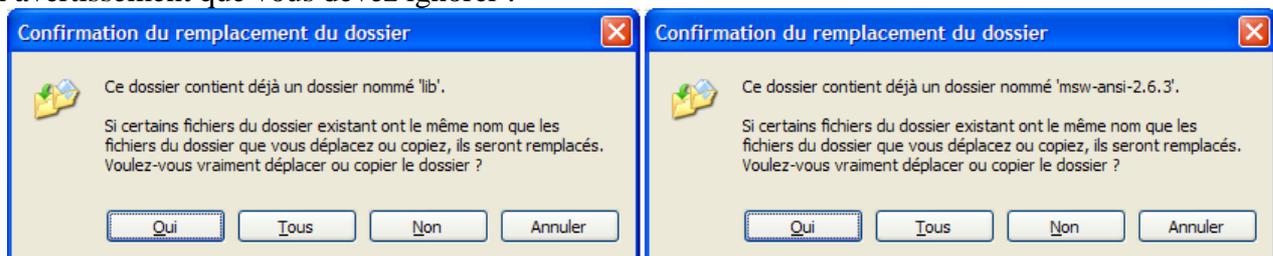
Il faut d'abord créer le répertoire dans lequel nous allons installer les bibliothèques. Pour cet exemple, nous allons créer le répertoire « **C:\wx263\** ».

Il faut également créer un sous-répertoire « **include** » dans ce répertoire d'installation.

## 4) Copie des fichiers

- Copier le contenu du sous-répertoire « **\packinclude\include\wxWidgets-2.6.3\** » qui se trouve dans le DevPack des en-têtes, et le coller dans le répertoire créé ci-dessus. Notre sous-répertoire « **include** » doit donc maintenant contenir un sous-répertoire « **wx** », ainsi qu'une ribambelle de fichiers « **\*.pch** »

- Copie le sous-répertoire « **lib** » qui se trouve dans chaque DevPack de bibliothèque compilée, et collez-le dans le répertoire d'installation. Normalement, si vous en installez plusieurs, Windows doit vous afficher un avertissement que vous devez ignorer :



- Si vous installez les DevPacks de bibliothèques dynamiques, le dossier « **lib** » contient les dll qu'il faut déplacer dans le répertoire « **system32** » de Windows. Pour ceux qui ont peur d'encombrer leur dossier « **system32** », vous pouvez, par exemple, créer un 3<sup>ème</sup> dossier sous « **C:\wx263\** » et vous le nommez « **D11** ». Copiez les dll dans ce dossier, et ajoutez ce chemin à la variable système « **PATH** », de façon à ce que Windows sache où les trouver.

## 5) L'installation des DevPacks est terminée.

Nous allons maintenant passer à leur utilisation en créant un petit projet basic sous Code::Blocks.

## Utilisation des DevPacks wxWidgets 2.6.3 avec Code::Blocks

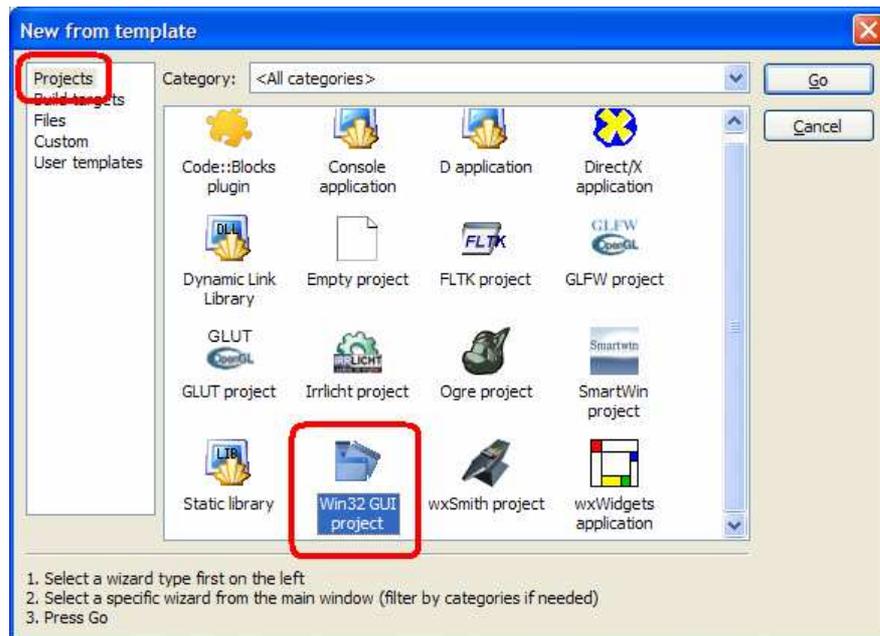
Afin d'être sûr d'obtenir les mêmes boîtes de dialogue que moi, je vous conseille de télécharger et d'installer la dernière « Nightly Build » (la mienne est celle du 23/08/2006).

Nous allons créer une petite application wxWidgets qui pourra vous servir de Template par la suite pour vos propres applications.

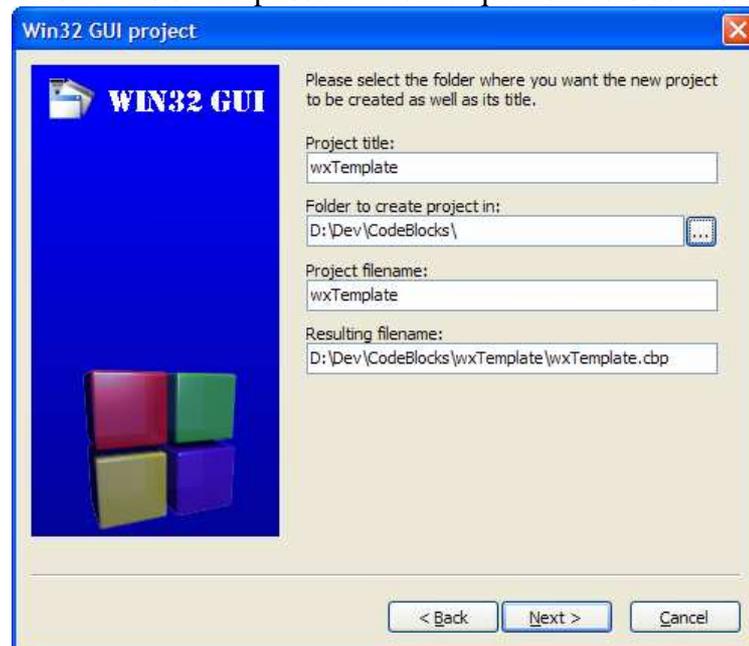
### 6) Démarrer Code::Blocks.

Nous allons partir d'un projet Win32 classique. Cela va nous permettre de voir quelles sont les options à donner au compilateur afin de pouvoir créer une application wxWidgets.

Pour commencer, nous allons créer un nouveau projet « **Win32 GUI project** ».



Après la page de bienvenue, l'assistant nous demande le nom, et le chemin du dossier dans lequel nous voulons créer ce nouveau projet. Pour ma part, j'appelle ce projet « wxTemplate », car il me servira par la suite de template pour toute nouvelle application wxWidgets. A noter qu'il suffit de remplir les deux premiers champs. Les deux suivants se remplissent automatiquement lors de la saisie.



La page suivante permet de régler les options de base pour les différentes configurations que contiendra notre projet. Pour l'instant, laissons les options par défaut qui nous sont proposées, nous les modifierons éventuellement par la suite. Voici ce que ça donne :



Voilà, c'est terminé, nous avons créé notre projet. Si vous regardez dans le dossier de ce projet, vous constaterez qu'il contient le fichier du projet (**wxTemplate.cbp**), et un fichier source (**main.cpp**). Si vous éditez le fichier **main.cpp**, vous remarquerez qu'il contient déjà du code pour créer une fenêtre en pur **Win32 API**. Nous allons donc tout supprimer afin d'obtenir un fichier vide pour pouvoir insérer notre code wxWidgets.

## 7) Création de la classe de base

Nous allons maintenant entrer le code minimum pour une application wxWidgets. En effet, dans toute application créée avec ce Framework, nous devons commencer par créer une classe dérivée de la classe wxApp. Cette classe devra au minimum contenir une fonction nommée « **OnInit** » et retournant un booléen pour indiquer si nous avons réussi à initialiser notre application ou non.

Nous n'allons pas pour l'instant créer de fenêtre, mais nous allons simplement afficher une « **MessageBox** » et terminer notre programme.

A ce propos, je vous ai dit ci-dessus que la fonction « **OnInit** » devait retourner un booléen.

- Si cette valeur est **true**, cela va indiquer à wxWidgets que nous avons correctement initialisé notre application, et que l'exécution peut continuer.
- Si cette valeur est **false**, c'est l'inverse : l'exécution se termine.

Pour cette première application, nous allons renvoyer false. En effet, après avoir affiché notre boîte de message, si nous indiquons à wxWidgets que tout est OK, notre programme va alors commencer la boucle habituelle (le Messages Loop) que toute application Win32 doit contenir. Comme nous n'avons pas encore de fenêtre à afficher, nous ne disposerons d'aucun moyen (à part le gestionnaire des tâches) pour terminer notre programme.

Après avoir créé notre classe, nous allons indiquer à wxWidgets qu'il s'agit de la classe de base de notre application. Nous disposons pour ce faire d'une macro qui prend en paramètre le nom de notre classe. Cette macro va se charger de créer la boucle des messages pour nous.

Vous noterez également, dans le code ci-après, la présence d'une ligne « **include <wx/wx.h>** »

Il s'agit du fichier minimum à inclure à tout code source basé sur wxWidgets. Il contient les définitions des classes et fonctions principales wxWidgets.

Voici donc ce que devra contenir le fichier `main.cpp` :

```
// Inclusion du fichier wx.h qui contient les définitions
// des classes et fonctions principales wxWidgets
#include <wx/wx.h>
// La définition de notre classe d'application
class MyApp : public wxApp
{
public:
    virtual bool OnInit();
};

// Indiquons à wxWidgets qu'il s'agit de notre classe d'application
// Et qu'elle devra servir de point d'entrée de l'exécutable.
IMPLEMENT_APP(MyApp);

bool MyApp::OnInit()
{
    // La fonction MessageBox de wxWidgets
    wxMessageBox("Démarrage de l'application !", "Génial", wxICON_INFORMATION);
    return false; // Pour ne pas bloquer l'exécution de l'application
}
```

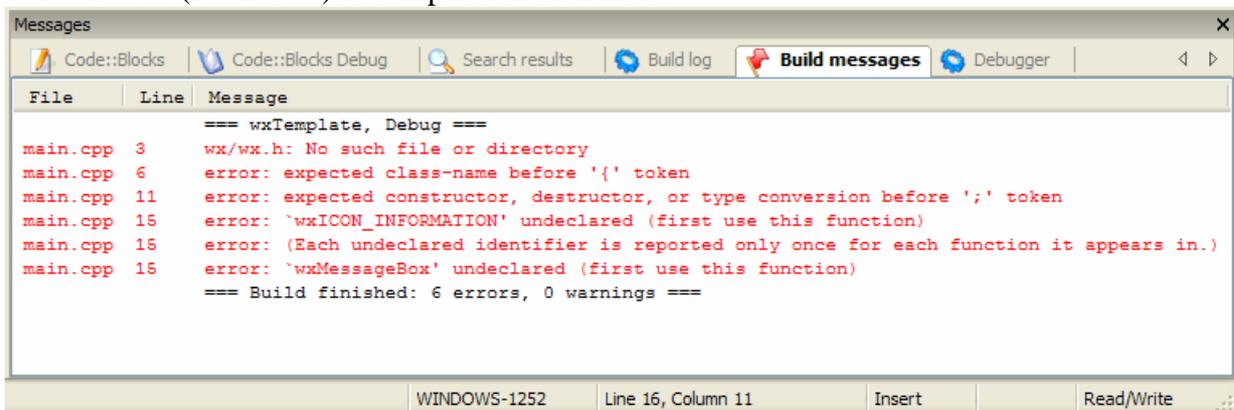
Voilà. C'est tout.

Il ne reste plus qu'à compiler pour voir ce que ça donne. Je vous laisse apprécier.

### 8) Les répertoires et options de compilation.

Et oui, ça ne donne rien d'autre qu'une série de messages d'erreurs. En effet, nous n'avons pas encore indiqué à Code::Blocks où nous avons placé les fichiers wxWidgets. C'est la partie la plus délicate. Heureusement, vous n'aurez pas à le refaire par la suite, car Code::Blocks vous donne la possibilité de vous servir d'un projet existant (et fonctionnant) comme Template pour une autre application (d'où le nom de projet wxTemplate, astucieux, non ?)

Voici le résultat de (l'échec de) la compilation chez moi :



The screenshot shows the 'Messages' window in Code::Blocks. The window title is 'Messages' and it has several tabs: 'Code::Blocks', 'Code::Blocks Debug', 'Search results', 'Build log', 'Build messages', and 'Debugger'. The 'Build messages' tab is active, displaying the following error messages:

File	Line	Message
		=== wxTemplate, Debug ===
main.cpp	3	wx/wx.h: No such file or directory
main.cpp	6	error: expected class-name before '{' token
main.cpp	11	error: expected constructor, destructor, or type conversion before ';' token
main.cpp	15	error: `wxICON_INFORMATION' undeclared (first use this function)
main.cpp	15	error: (Each undeclared identifier is reported only once for each function it appears in.)
main.cpp	15	error: `wxMessageBox' undeclared (first use this function)
		=== Build finished: 6 errors, 0 warnings ===

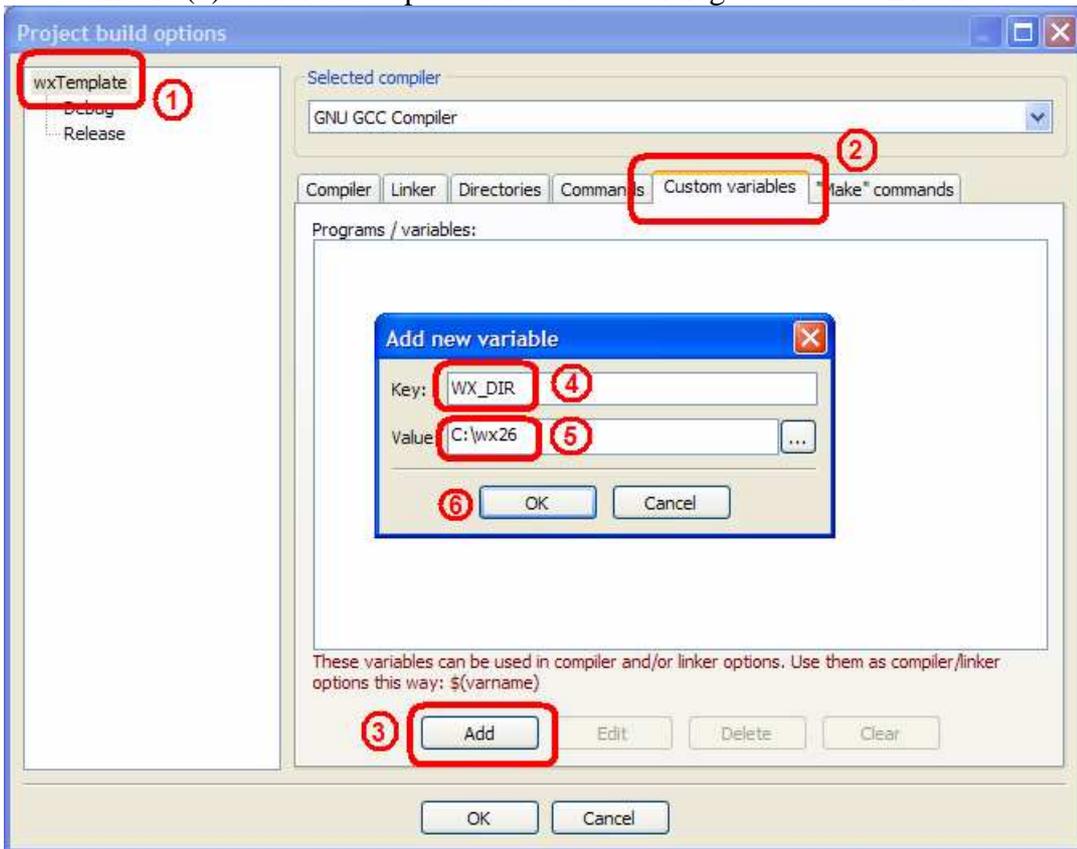
The status bar at the bottom of the window shows 'WINDOWS-1252', 'Line 16, Column 11', 'Insert', and 'Read/Write'.

La première ligne nous dit que le compilateur ne trouve pas le fichier « `wx/wx.h` ». En effet, il faut lui indiquer que nous avons placé les « `includes` » dans le répertoire « `c:\wx26\include` ».

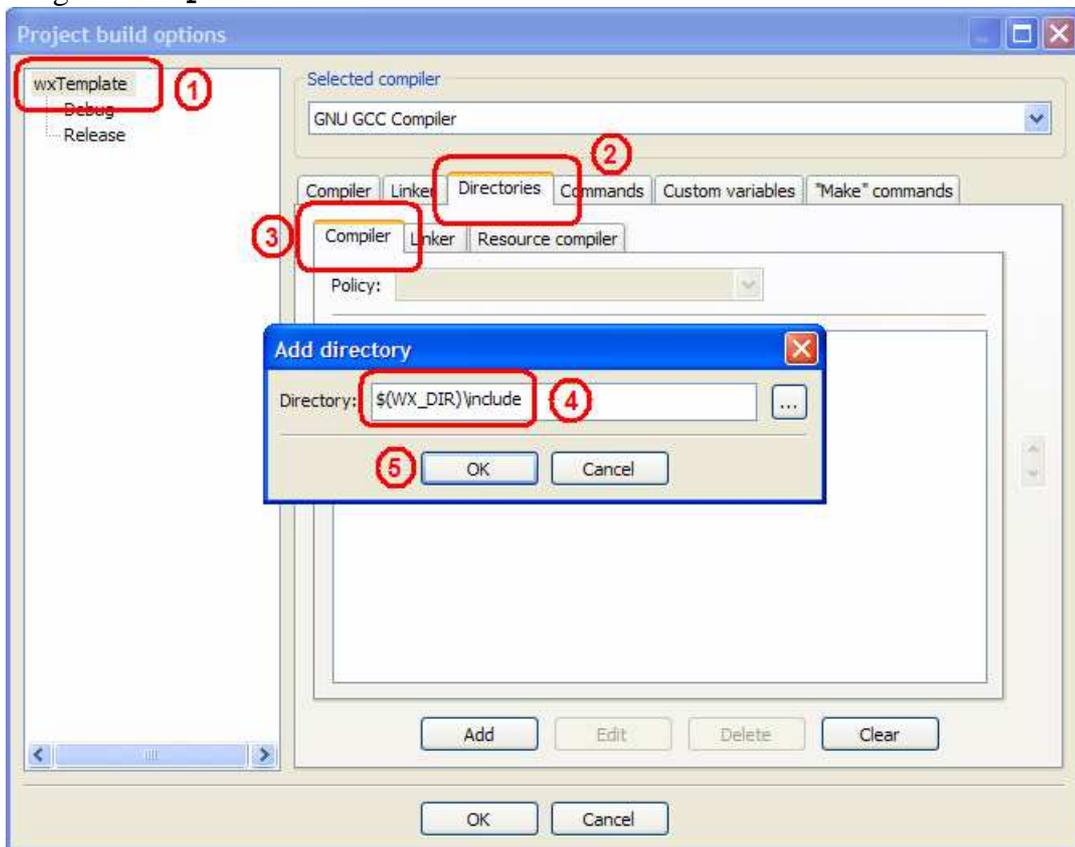
Par la suite, nous verrons qu'il nous faudra aussi lui indiquer que nous avons placé les fichiers lib dans un autre sous-répertoire de « `c:\wx26\` ». Nous allons donc créer une variable globale à notre projet, que nous appellerons « `WX26_DIR` », et qui contiendra le chemin de base de wxWidgets, c'est-à-dire « `c:\wx26\` ».

Pour ce faire, il faut aller dans la boîte d'options de compilation du projet. Pour l'afficher, il faut aller dans le menu « `Project` », « `Build options` ». Comme il s'agit d'une variable globale à notre projet, et non à l'une des configurations, il faut sélectionner l'entrée « `wxTemplate` » dans l'arbre (1), puis l'onglet « `Custom variables` » (2). La liste des variables doit pour l'instant être vide. Nous allons donc ajouter notre variable (bouton « `Add` ») (3) Entrez ensuite le nom de la variable, soit « `WX_DIR` » (4) et sa valeur (5).

Attention : il ne faut pas ajouter le dernier antislash au chemin, il risquerait de faire doublon lors de l'utilisation de cette variable (je vous le rappellerai au moment venu). Il ne nous reste plus qu'à valider cette variable (6). Voici ces étapes sur la boîte de dialogue de Code::Blocks.



De même, comme les fichiers « **include** » seront toujours placés au même endroit, nous pouvons indiquer leur emplacement dans les options du projet. Cela se passe dans l'onglet « **Directories** », et dans le sous-onglet « **Compiler** ».



Vous noterez au passage la façon dont j'ai donné le nom du répertoire au compilateur :

« `$(WX_DIR)\include` ». C'est pour cette raison que je vous ai indiqué, précédemment, de ne pas mettre le dernier antislash de la variable `WX_DIR`. En effet, il aurait été doublé par celui précédant « `include` » dans ce chemin (cela aurait donné : `C:\wx26\include`).

Voilà, notre compilateur sait maintenant où aller chercher les headers de `wxWidgets`. Il ne nous reste plus qu'à fermer la boîte d'options, et à relancer la compilation.

Surprise, ça ne marche toujours pas ! C'est même pire : nous obtenons beaucoup plus d'erreurs que la première fois. Si l'on regarde la première erreur, elle concerne encore un header non trouvé : le fichier « `wx/setup.h` ». Pour faire simple, ce fichier est celui dans lequel sont données toutes les options qui ont servi à compiler `wxWidgets`. Ainsi, certaines définitions ne sont pas les mêmes si l'on programme en UNICODE ou en ANSI. Il faudra donc inclure le bon suivant vos préférences de programmation.

### 9) Différentes configurations de compilation.

Pour commencer, nous allons nous occuper de programmer en « ANSI », et en utilisant les libs dynamiques. Notre projet contient actuellement deux configurations : « `Debug` » et « `Release` ». Nous allons donc les renommer en « `Debug_Ansi_Dynamic` » et « `Release_Ansi_Dynamic` ». Pour ce faire, il faut aller dans la page de propriétés du projet : menu « `Project` », « `Properties` ». Ne nous occupons pas pour l'instant du premier onglet, et allons tout de suite dans l'onglet « `Targets` ». Sélectionnez chacune des configurations, dans la liste de gauche, et renommez-les comme indiqué ci-dessus. Vous remarquerez également, dans la partie de droite, que les valeurs de « `Output filename` » et « `Objects output dir` » ne sont pas les mêmes suivant la configuration qui est sélectionnée. Il faut donc également les modifier, pour qu'elles n'interfèrent pas avec celles des configurations que nous serons amenés à créer par la suite. Il faut donc mettre les valeurs suivantes (je raccourci volontairement les noms des dossiers, mais vous faites comme vous voulez :

Configuration	Output filename	Objects output dir
<code>Debug_Ansi_Dynamic</code>	<code>bin\deb_ansi_dll\wxTemplate.exe</code>	<code>obj\deb_ansi_dll</code>
<code>Release_Ansi_Dynamic</code>	<code>bin\rel_ansi_dll\wxTemplate.exe</code>	<code>obj\rel_ansi_dll</code>

Comme nous avons modifié les emplacements des fichiers intermédiaires et du fichier final, et pendant que nous y pensons, nous pouvons supprimer les éventuels sous-répertoires « `debug` » et « `release` » qui se trouvent dans les dossiers « `bin` » et « `obj` » de notre projet (à noter qu'ils peuvent ne pas être présents si vous n'avez pas essayé de compiler avec ces configurations).

Revenons à notre boîte d'options : nous allons maintenant expliquer au compilateur où il peut trouver le fichier « `setup.h` » correspondant à chacune de nos configurations. Pour commencer, sélectionnez la config « `Debug_Ansi_Dynamic` », et cliquez sur le bouton « `Build options` ». Nous retombons sur la boîte d'options de compilation que nous avons eue la première fois.

Nous allons créer une variable propre à chaque configuration, que nous appellerons « `WX_CFG` ». Nous allons lui donner la valeur « `msw-ansi-2.6.3` ». Pourquoi cette valeur ? Si vous jetez un coup d'œil dans l'arborescence du dossier « `C:\wx26\` », et que vous avez installé tous les DevPacks, vous vous apercevrez que l'on y trouve les dossiers suivants (les couleurs vous permettront de mieux voir les concordances):

```
C:\wx26\include\wx\msw-ansi-2.6.3\  
C:\wx26\include\wx\msw-unicode-2.6.3\  
C:\wx26\lib\msw-ansi-2.6.3\Dynamic\  
C:\wx26\lib\msw-ansi-2.6.3\Static\  
C:\wx26\lib\msw-unicode-2.6.3\Dynamic\  
C:\wx26\lib\msw-unicode-2.6.3\Static\
```

De plus, si vous fouillez un peu plus, vous verrez que les deux premiers dossiers contiennent chacun un dossier « **wx** » dans lequel se trouve le fichier « **setup.h** » que notre compilateur recherche. Il faut donc que l'on ajoute ce dossier à la liste des répertoires dans lesquels notre compilateur doit chercher les headers.

Pour cela (nous sommes toujours sur la boîte d'options de compilation) :

- Sélectionnez la configuration « **Debug\_Ansi\_Dynamic** ».
- Onglet « **Custom variables** » (la liste doit être vide)
- Ajoutez la variable « **WX\_CFG** » et donnez-lui la valeur « **msw-ansi-2.6.3** »
- Onglet « **Directories** », sous-onglet « **Compiler** » (la liste est également vide)
- Ajoutez l'entrée « **\$(WX\_DIR)\include\wx\\$(WX\_CFG)** » (ce qui donnera au compilateur, pour cette configuration, la valeur finale « **C:\wx26\include\wx\msw-ansi-2.6.3** »)

Il faut faire la même chose pour la deuxième configuration. En effet, les seules différences entre ces deux configurations sont les infos de débogage qui sont intégrées à la première, ainsi que le fait qu'elle démarrera en mode console.

- Sélectionnez la configuration « **Release\_Ansi\_Dynamic** »
- Onglet « **Custom variables** » (encore une liste vide)
- Ajouter la variable « **WX\_CFG** » avec pour valeur « **msw-ansi-2.6.3** »
- Onglet « **Directories** », sous-onglet « **Compiler** » (une liste vide de plus...)
- Ajouter « **\$(WX\_DIR)\include\wx\\$(WX\_CFG)** »

Ok. Maintenant, le compilateur sait où trouver le fichier « **wx/setup.h** ». Nous pouvons donc fermer les boîtes d'options qui sont ouvertes, et lancer la compilation.

Vous y avez cru ? Et bien non, ça ne marche toujours pas... C'est même de pire en pire, au niveau des erreurs. Par contre, ce qui a évolué, c'est que ce ne sont plus des erreurs de compilation, mais d'édition de liens. La compilation du fichier « **main.cpp** » a bien été effectuée. Pour vous en assurer, jetez un coup d'œil dans le répertoire « **obj\deb\_ansi\_dll\** » du projet. Vous verrez qu'il contient un fichier « **main.o** », qui est le fichier objet de la compilation de « **main.cpp** ».

Nous allons maintenant indiquer au compilateur (euh, non, au linker en fait) qu'il doit lier notre exécutable aux libs wxWidgets. Il va donc dans un premier temps lui indiquer où se trouvent ces libs (nous, on le sait, elles se trouvent dans les 4 derniers répertoires de la liste colorée que je vous ai donné précédemment).

- Ouvrez la boîte d'options de compilation (menu **Project, Build options**).
- Sélectionnez la première configuration (« **Debug\_Ansi\_Dynamic** »)
- Onglet « **Directories** », sous-onglet « **linker** » (tiens, encore une liste vide).
- Ajoutez le répertoire « **\$(WX\_DIR)\lib\\$(WX\_CFG)\Dynamic** », ce qui donnera au linker comme valeur finale : « **C:\wx26\lib\msw-ansi-2.6.3\Dynamic** ».

Répétez cette opération avec les mêmes valeurs pour la configuration « **Release\_Ansi\_Dynamic** » (oui, je sais, il s'agit encore d'une liste vide...).

Il nous reste à indiquer quelles sont les fichiers auxquels le linker doit lier notre exécutable. Pour une application de base, les seules dll qui nous seront utiles sont « **wx\_base26\_gcc\_custom.dll** » et « **wxmsw26\_core\_gcc\_custom.dll** ». Les informations concernant ces dll sont respectivement contenues dans les libs « **libwx\_base-2.6.dll.a** » et « **libwx\_msw\_core-2.6.dll.a** ». Pour indiquer au linker de les utiliser, toujours dans la boîte d'options de compilation et pour chaque configuration :

- Sélectionnez l'onglet « **linker** »
- Ajoutez dans la liste « **Link libraries** » (vide, elle aussi) les valeurs « **wx\_base-2.6** » et « **wx\_msw\_core-2.6** ».

Encore une petite chose : il nous faut indiquer au compilateur que nous utilisons les libs dynamiques, et non statiques. Cela se fait en définissant la variable « **WXUSINGDLL** » dans les options du compilateur.

Toujours dans la boîte d'options de compilation, et pour les deux configurations :

- Onglet « **Compiler** », sous-onglet « **#defines** »

- Ajoutez la valeur « **WXUSINGDLL** » à la liste qui pour l'instant est vide.

Vous pouvez maintenant fermer la boîte d'options de compilation pour retenter de compiler notre projet.

Normalement, cette fois-ci, ça devrait marcher (malgré les quelques infos qui sont apparues à la compilation). Si toutefois la compilation échoue, essayez de faire un nettoyage (menu « **Build** », « **Clean workspace** »). En effet, il se peut que quelques fichiers mal compilés soient présents dans les sous-répertoires du projet.

Au final, si vous exécutez votre application, vous devriez obtenir une jolie MessageBox, ainsi qu'une fenêtre « **Console** » pour la configuration « **Debug\_Ansi\_Dynamic** ».

Malgré le fait que la compilation soit Ok, il nous reste quand même quelques instructions à donner au compilateur, pour obtenir un exécutable plus propre, ainsi que pour gagner du temps à la compilation. En effet, les libs wxWidgets sont tellement complètes en termes de diversité des classes que la compilation d'une application les utilisant devient vite longue.

Pour optimiser le temps de compilation, nous allons utiliser les en-têtes précompilées. Cette technique consiste à compiler une partie des fichiers headers dont votre programme a besoin. Cette compilation peut être un peu longue, mais elle n'est à faire qu'une seule fois. Par la suite, la compilation de vos fichiers sources fera directement appel à ces headers précompilés, et, vous le constaterez par vous-même, vous fera gagner du temps.

Nous allons donc créer le fichier en-tête qui nous permettra de générer ces headers précompilés. Pour ce faire, nous allons utiliser l'assistant de Code::Blocks (menu « **File** », « **New** », dans la liste de gauche, sélectionnez « **Files** », et dans la liste de droite « **C/C++ Header** »). Passez la page de bienvenue de l'assistant, et, dans la page qui suit, nommez ce nouveau fichier « **wx\_pch.h** ». Vous remarquerez que le champ « **Header guard block** » s'est automatiquement rempli avec la valeur « **WX\_PCH\_H\_INCLUDED** ». (Je suppose que vous connaissez les bases du C++, je n'ai donc pas besoin de vous expliquer le pourquoi de cette manœuvre). Veillez à ce que la checkbox « **Add File to active project** » soit bien cochée, et qu'une configuration soit bien sélectionnée dans la combobox associée.

Petite astuce : Si vous saisissez uniquement « **wx\_pch.h** » dans le champ « **Filename** », Code::Blocks vous affichera une erreur. Il faut saisir le chemin complet du fichier, ou se servir du bouton « **Parcourir** » (...) à droite de ce champ.

Notre fichier est maintenant créé. Il ne nous reste plus qu'à lui demander d'inclure les headers précompilés de wxWidgets. Pour cela, nous disposons d'un header spécial : « **wxprec.h** ». C'est lui qui va déterminer les autres fichiers à inclure pour les libs wxWidgets. Il va également activer une directive de compilation conditionnelle : « **WX\_PRECOMP** », pour indiquer que les headers précompilés sont bien supportés par notre compilateur. Nous allons également, de notre côté, définir une variable de ce type pour pouvoir « **forcer** » l'utilisation de cette fonctionnalité. Nous l'appellerons « **USE\_PCH** ».

Voici donc le contenu de notre fichier « wx\_pch.h » :

```
#ifndef WX_PCH_H_INCLUDED
#define WX_PCH_H_INCLUDED

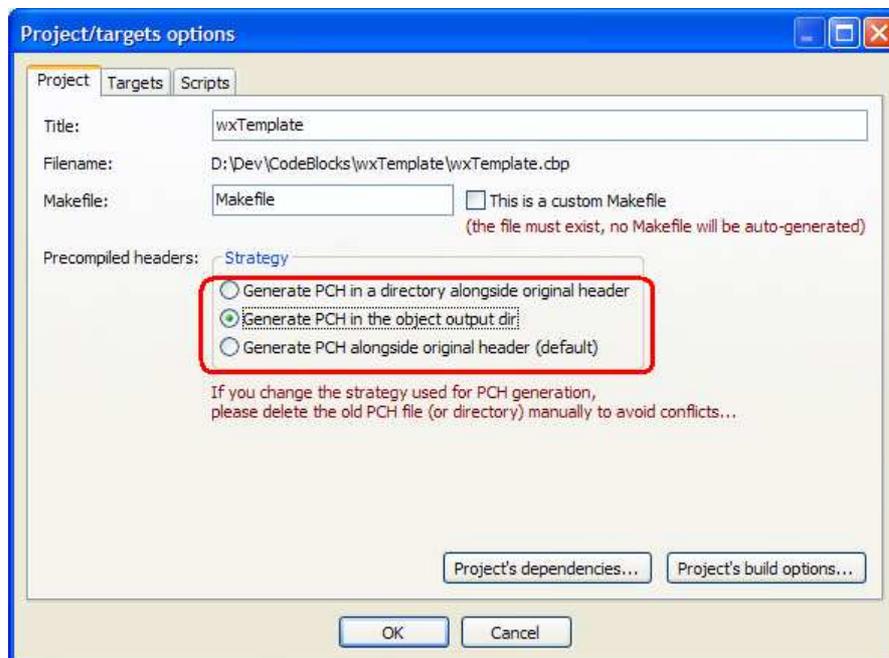
// Si nous voulons forcer les headers précompilés, nous pouvons d'ores
// et déjà définir la variable WX_PRECOM
#if ( defined(USE_PCH) && !defined(WX_PRECOMP) )
#define WX_PRECOMP
#endif // USE_PCH

// Les headers précompilés de base pour wxWidgets
#include <wx/wxprec.h>
#ifndef WX_PRECOMP
#include <wx/wx.h>
#endif

#endif // WX_PCH_H_INCLUDED
```

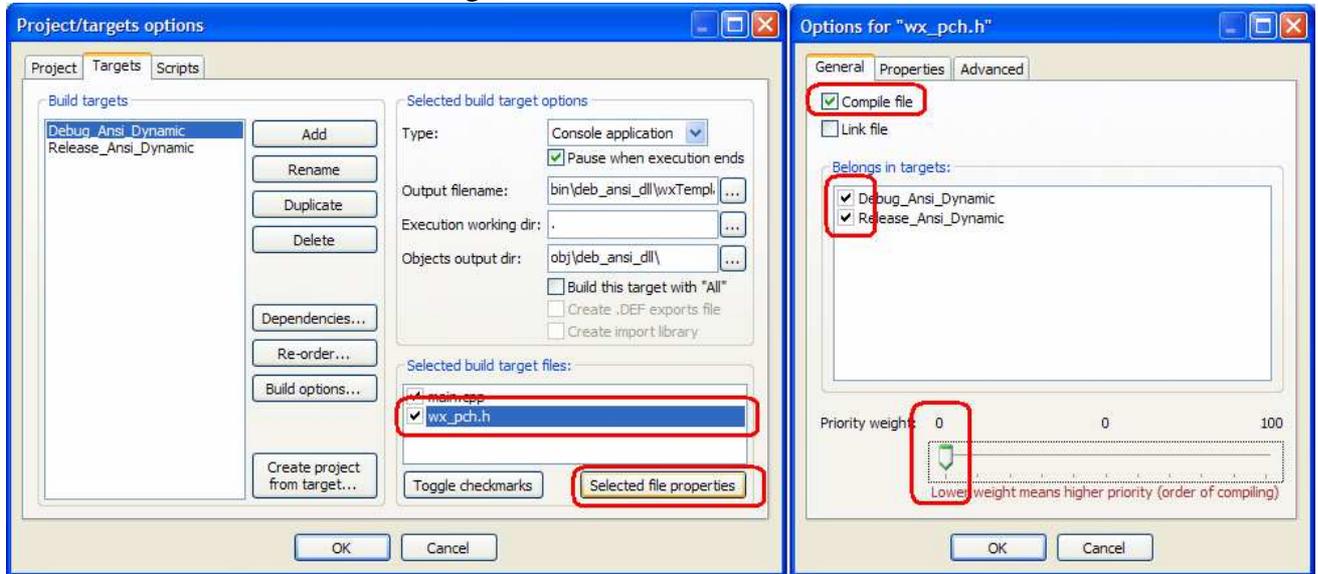
Il nous faut maintenant expliquer à Code::Blocks qu'il doit compiler ce fichier avec chacune de nos configurations ; et nous devons également ajouter notre variable « USE\_PCH » dans les options du compilateur.

Ouvrez la boîte de dialogue « **Project options** » (menu « **Project** », « **Properties** »). Dans le premier onglet (« **Project** »), vous trouverez une section consacrée aux headers précompilés. C'est ici que l'on va indiquer au compilateur à quel endroit il doit placer le résultat de cette précompilation. Pour ma part, je préfère ne pas laisser l'option par défaut qui dit de générer les headers précompilés au même endroit que le fichier header lui-même. Je préfère les placer dans le répertoire « objet », avec les autres fichiers compilés.



Passez ensuite dans l'onglet « **Targets** ». Nous allons nous assurer que le fichier « wx\_pch.h » va bien être compilé (ce qui n'est normalement pas le cas pour un fichier header ; cette option n'est donc pas activée par défaut). Pour cela : sélectionnez le fichier « wx\_pch.h » dans la liste des fichiers en bas à droite, et pressez le bouton « **Selected file properties** ». Dans la boîte qui apparaît, il faut donc cocher la case « **Compile file** ». Nous voulons que cela se fasse pour nos deux configurations, il faut donc cocher les deux cases correspondantes dans la liste « **Belongs in target** ». Nous allons également dire à Code::Blocks de compiler ce fichier en premier. Cela se fait en faisant glisser le curseur du bas vers la gauche.

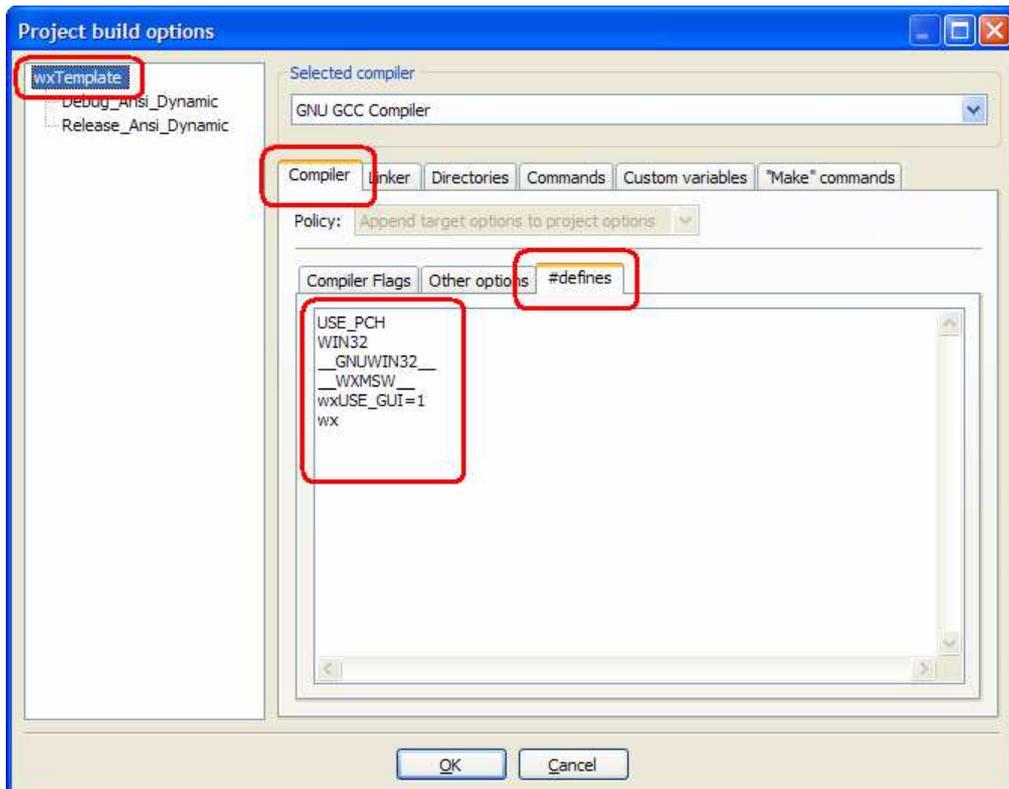
Voici le résultat de cette boîte de dialogue :



Bien, maintenant, il nous reste quelques petits réglages à faire, et tout devrait être Ok pour cette première partie de notre application. Souvenez-vous, je vous ai dit précédemment que nous allons définir une variable « **USE\_PCH** ». Ce n'est pas la seule. Nous devons également définir une variable « **WIN32** », permettant de n'utiliser que les fonctions valides pour cette plateforme ; une variable « **\_\_GNUWIN32\_\_** » pour indiquer que l'on utilise le compilateur **GNU** (MinGW) ; une variable « **\_\_WXMSW\_\_** » pour indiquer que l'on va utiliser la version Microsoft Windows de wxWidgets ; une variable « **wxUSE\_GUI=1** » pour indiquer à wxWidgets que nous allons utiliser des fenêtres, et des contrôles ; et une variable « **wx** » qui sert ... je ne sais pas à quoi, mais il faut qu'elle soit là.

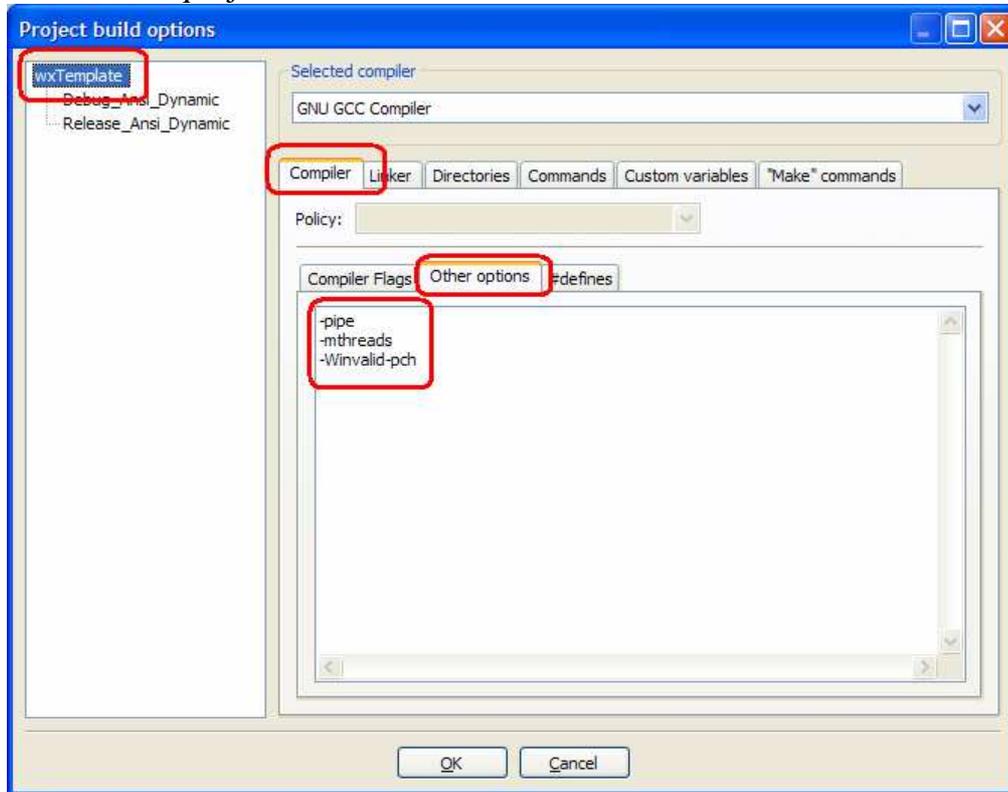
De même, pour la configuration « **Debug\_Anisi\_Dynamic** », nous allons ajouter une variable « **DEBUG** » qui nous servira à éventuellement ajouter des actions (des wxMessageBox, par exemple) uniquement en mode debug.

Pour cela, ouvrez la boîte de dialogue « **Build Options** » depuis le menu « **Project** », sélectionnez tout d'abord le projet « **wxTemplate** », l'onglet « **Compiler** », et le sous-onglet « **#defines** ». Il suffit de rajouter dans la liste les 6 variables communes aux deux configurations dont je viens de vous parler, comme ceci :



Ensuite, sélectionnez la configuration « **Debug\_Ansi\_Dynamic** ». Toujours dans le même onglet/sous-onglet (vous devez déjà avoir une variable « **WXUSINGDLL** »), ajoutez la variable propre à cette configuration : « **DEBUG** » (je vous épargne la capture d'écran cette fois-ci).

Il reste également à ajouter quelques options au compilateur : re-sélectionnez le projet « **wxTemplate** », l'onglet « **Compiler** », et le sous-onglet « **Other options** », et entrez les trois valeurs comme ci-dessous. Je ne sais pas trop à quoi elles servent (à part la deuxième qui sert à activer le multi-threading pour notre application). Pour me justifier de cette lacune, je peux seulement vous dire que je ne suis pas un programmeur professionnel, et que je n'ai jamais suivi de cours c++. Je sais, ce n'est pas une excuse valable, mais c'est la seule que j'ai trouvée...



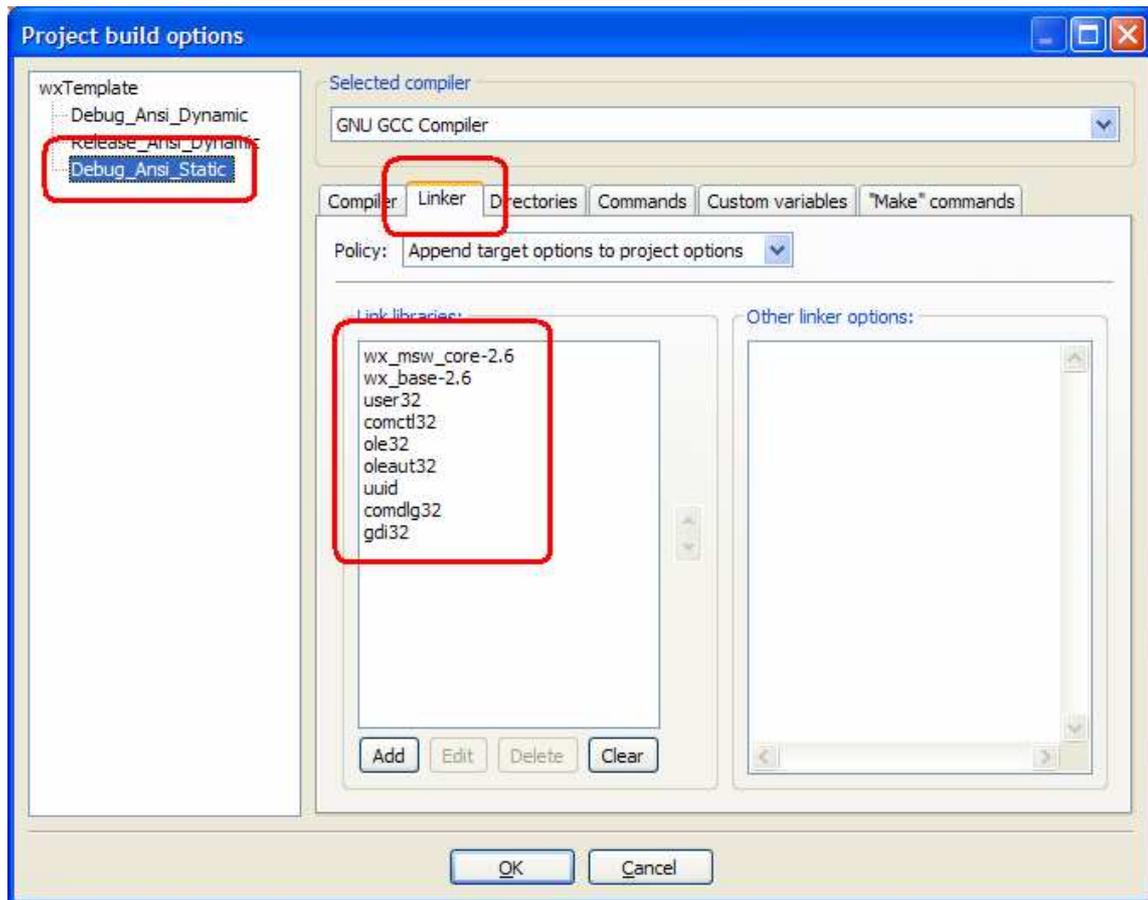
Voilà, c'est maintenant terminé pour les réglages concernant nos deux configurations. Vous pouvez fermer la boîte de dialogue. Pour tout remettre à plat, je vous conseille de faire un « **Rebuild** » afin que tous les fichiers compilés prennent en compte les dernières options. Nous n'avons pour l'instant pas fait de jolie interface, mais nous avons un projet qui marche.

## 10) Ajouter des configurations

Nous avons jusqu'à maintenant travaillé avec les libs Ansi Dynamiques. Nous allons maintenant voir comment créer les configurations « **Debug** » et « **Release** » pour les libs Ansi Statiques. Pour ne pas avoir à tout refaire, nous allons simplement partir d'une copie des configurations dont nous disposons actuellement. Pour cela, ouvrez la boîte de dialogue « **Project/targets options** » en allant dans le menu « **Project** », « **Properties** ». Dans l'onglet « **Targets** », sélectionnez la configuration « **Debug\_Ansi\_Dynamique** », et faites-en une copie par l'intermédiaire du bouton « **Duplicate** ». Nommez cette nouvelle configuration « **Debug\_Ansi\_Static** », et validez. Assurez vous que cette nouvelle configuration soit bien sélectionnée, et modifiez les valeurs des champs « **Output filename** » et « **Objects output dir** », comme nous l'avons fait au chapitre 9 pour les deux premières configurations :

- « **Output filename** » deviendra « **bin\deb\_ansi\_stat\wxTemplate.exe** ».
- « **Objects output dir** » deviendra « **obj\deb\_ansi\_stat\** ».
- Il faut maintenant indiquer au linker d'utiliser les libs statiques à la place des libs dynamiques. C'est dans la boîte de dialogue « **Project build options** » que ça se passe (souvenez-vous, au chapitre 9, la liste des répertoires avec un code de couleurs).

- Il suffit donc, pour la configuration « **Debug\_Ansi\_Static** », dans l'onglet « **Directories** », le sous-onglet « **Linker** », de modifier la valeur existante qui est (du fait qu'elle ait été copiée depuis la configuration « **Debug\_Ansi\_Dynamic** ») «  $\$(WX\_DIR)\lib\$(WX\_CFG)\Dynamic\$  », en la remplaçant par «  $\$(WX\_DIR)\lib\$(WX\_CFG)\Static\$  », et le tour est joué. Il faut également que l'on enlève la directive « **WXUSINGDLL** » que nous avons placé dans les options du compilateur (onglet « **Compiler** », sous-onglet « **#defines** »).
- Il reste néanmoins un petit détail à régler : jusqu'à maintenant, nous avons travaillé avec des libs dynamiques. Elles faisaient donc référence à des fichiers DLL, qui eux étaient liés aux DLL Windows. Pour les libs statiques, il en est tout autrement. Elles ne font pas référence à des dll, mais leur code compilé est ajouté à l'exécutable. Il faut donc que nous ajoutions les liens vers les dll Windows nous même pour que la compilation aboutisse. Voici donc la liste des libs à ajouter dans la liste, dans l'onglet « **Linker** ». Je vous conseille de les ajouter dans cet ordre, sans quoi, vous risqueriez d'obtenir des problèmes au link (Il faut d'abord que le lien soit fait avec les libs wxWidgets, puis avec les dll Windows pour que ça marche correctement ; il faut également que la lib « **wx\_msw\_core-2.6** » soit placée avant la lib « **wx\_base-2.6** »).



- Voilà, notre configuration « **Debug\_Ansi\_Static** » est maintenant opérationnelle. Vous pouvez essayer de la compiler, pour vérifier.
- Je vous laisse faire la même manipulation pour créer la configuration « **Release\_Ansi\_Static** ».
- Faites une copie de la config « **Release\_Ansi\_Dynamic** »
- « **Output filename** » devient « **bin\rel\_ansi\_stat\wxTemplate.exe** »
- « **Output objects dir** » devient « **obj\rel\_ansi\_stat** »
- Le répertoire contenant les libs devient «  $\$(WX\_DIR)\lib\$(WX\_CFG)\Static$  »
- Suppression de la directive « **WXUSINGDLL** »
- Ajout des libs en respectant l'ordre de la capture d'écran.

Et voilà, nous en avons maintenant terminé avec les libs Ansi.  
Nous allons maintenant passer aux libs Unicode.

## 11) Les libs Unicode

Nous allons procéder de la même manière pour ajouter les configurations ayant rapport aux libs Unicode, à quelques détails près.

Tout d'abord, nous allons créer la première configuration, à savoir « **Debug\_Unicode\_Dynamic** », en dupliquant la configuration « **Debug\_Ansi\_Dynamic** ».

- « **Output filename** » devient « **bin\deb\_unic\dll\wxTemplate.exe** »
- « **Objects output dir** » devient « **obj\deb\_unic\_dll** ».
- La définition du répertoire des libs ne va pas changer. C'est, par contre, la variable « **WX\_CFG** » que nous allons modifier cette fois-ci (onglet « **Custom variables** »). Elle va devenir (toujours suivant le schéma de la liste en couleur du chapitre 9) « **msw-unicode-2.6.3** ».
- Nous laissons la directive « **WXUSINGDLL** », car nous recommençons avec des libs dynamiques, et nous n'ajoutons pas les libs des dll Windows.
- Par contre, si vous regardez dans le répertoire contenant les libs unicode wxWidgets, les noms des fichiers ne sont pas tout à fait les mêmes. Par exemple, le fichier « **libwx\_base-2.6.dll.a** » est devenu « **libwx\_baseu-2.6.dll.a** ». Le « **u** » (pour Unicode) a été ajouté à toutes les libs afin de bien savoir avec lesquelles on travaille. Qu'à cela ne tienne, nous allons donc modifier le nom des libs dans la boîte de dialogue « **Project build options** », en ajoutant ce « **u** » aux deux libs déjà présentes. « **wx\_base-2.6** » devient donc « **wx\_baseu-2.6** » et « **wx\_msw\_core-2.6** » devient « **wx\_mswu\_core-2.6** ».
- Il reste encore à indiquer au compilateur que nous travaillons avec les libs Unicode, en ajoutant la définition « **UNICODE** » dans la liste des « **#defines** ».

On ferme les boîtes de dialogue ouvertes, on compile notre nouvelle configuration.....et on obtient une erreur dans notre code (qui marchait pourtant bien jusqu'à maintenant).

Explication : La fonction `wxMessageBox` prend comme premier paramètre un « `const wxString` ». Avec les libs Ansi, les chaînes de caractères entre guillemets sont considérées comme de telles valeurs. En Unicode, il en est tout autrement (je ne rentrerais pas dans les détails, car je ne suis pas un spécialiste en Unicode). Pour palier à ce problème, les libs wxWidgets nous proposent une macro prenant en paramètre une chaîne de caractères classique, et la convertissant en « `const wxString` » quelque soit notre configuration. Du coup, il devient un peu plus fastidieux de saisir une chaîne de caractères, mais la compatibilité Ansi-Unicode est ainsi (presque) assurée. Cette macro s'utilise de la façon suivante :

```
_("Chaîne de caractères")
```

Ainsi, nous devons modifier les deux chaînes de caractères de notre `wxMessageBox`.

Ça ne sera pas tout. En effet, les chaînes de caractères Unicode n'acceptent pas les caractères accentués. Du coup, c'est raté pour le mot « **Démarrage** » et pour le titre « **Génial** ». Trois solutions s'offrent à nous :

- Réaliser des logiciels uniquement en langue anglaise (donc sans accents)
- Remplacer tous les caractères accentués par leurs équivalents classiques
- Remplacer tous les caractères accentués par leur valeur dans la table des caractères.

Vous avez deviné quelle solution nous allons choisir ? La troisième, évidemment ! Il nous faut donc récupérer le code du caractère « **é** ». Pour cela, lancez la table des caractères de Windows (**Démarrer, Tous les programmes, Accessoires, Outils système, Table des caractères**). Sélectionnez le caractère qui nous intéresse, et regardez les informations qui vous sont données sur ce caractère dans la barre d'état : « **U+00E9 : LETTRE MINUSCULE LATINE E ACCENT AIGU** ». C'est le « **U+00E9** » qui nous intéresse. Cela veut dire « Caractère numéro **0x00E9** (en hexadécimal) dans l'alphabet Unicode ». Très bien : Une petite conversion nous apprend que **0x00E9** équivaut à **351** en octal. Il nous suffit de remplacer les caractères « **é** » dans notre fonction par « **\351** », et le tour est joué. Notre fonction devient donc :

```
wxMessageBox(_("D\351marrage de l'application !"), _("G\351nial"), wxICON_INFORMATION);
```

C'est un peu bizarre et contraignant, surtout quand on en n'a pas l'habitude, mais par contre, c'est efficace : La compilation se fait sans problème : testez par vous-même !

Passons maintenant à la configuration « **Release\_Unicode\_Dynamic** ». Vous devez commencer à connaître la procédure à suivre :

- Dupliquer la configuration « **Release\_Ansi\_Dynamic** » en « **Release\_Unicode\_Dynamic** ».
- « **Output filename** » devient « **bin\rel\_unic\_dll\wxTemplate.exe** »
- « **Objects output dir** » devient « **obj\rel\_unic\_dll** »
- « **WX\_CFG** » devient « **msw-unicode-2.6.3** »
- « **wx\_base-2.6** » devient « **wx\_baseu-2.6** »
- « **wx\_msw\_core-2.6** » devient « **wx\_mswu\_core-2.6** »
- Ajout de « **UNICODE** » dans la liste des « **#defines** »

Pour la configuration « **Debug\_Unicode\_Static** » :

- Dupliquer la configuration « **Debug\_Ansi\_Static** » en « **Debug\_Unicode\_Static** »
- « **Output filename** » devient « **bin\deb\_unic\_stat\wxTemplate.exe** »
- « **Objects output dir** » devient « **obj\deb\_unic\_stat** »
- « **WX\_CFG** » devient « **msw\_unicode-2.6.3** »
- « **wx\_base-2.6** » devient « **wx\_baseu-2.6** »
- « **wx\_msw\_core-2.6** » devient « **wx\_mswu\_core-2.6** »
- Ajout de « **UNICODE** » dans le liste des « **#defines** »

Et enfin, pour la configuration « **Release\_Unicode\_Static** » :

- Dupliquer la configuration « **Release\_Ansi\_Static** » en « **Release\_Unicode\_Static** »
- « **Output filename** » devient « **bin\rel\_unic\_stat\wxTemplate.exe** »
- « **Objects output dir** » devient « **obj\rel\_unic\_stat** »
- « **WX\_CFG** » devient « **msw\_unicode-2.6.3** »
- « **wx\_base-2.6** » devient « **wx\_baseu-2.6** »
- « **wx\_msw\_core-2.6** » devient « **wx\_mswu\_core-2.6** »
- Ajout de « **UNICODE** » dans le liste des « **#defines** »

Et voilà, nous avons un projet wxWidgets sous Code::Blocks totalement opérationnel, permettant de compiler un programme avec huit configurations différentes.

Nous allons maintenant nous servir de ce projet pour créer un squelette d'application pour Code::Blocks. Ainsi, lorsque vous aurez besoin de créer un nouveau projet utilisant wxWidgets, vous n'aurez plus besoin de refaire toutes les manipulations précédentes.

## 12) Le nouveau Template sous Code::Blocks

Pour créer un template à partir d'un projet existant, il suffit d'aller dans le menu « **File** », et de sélectionner l'entrée « **Save project as user template** ». Un nom vous sera alors demandé pour votre template. Vous pourrez ensuite y avoir accès en choisissant « **User Templates** » dans la boîte de dialogue « **New project** » (voir la 1<sup>ère</sup> capture du chapitre 6).

## 13) C'est tout pour l'instant

Voilà, c'est fini (pour le moment en tout cas). La prochaine mise à jour de ce tutoriel concernera l'ajout de code pour créer une fenêtre, y ajouter des contrôles, et connecter ces contrôles aux fonctions de notre classe de fenêtre.

En attendant, vous pouvez retrouver le projet final de ce tutoriel sur ma page perso, à l'adresse : <http://xaviou.chez-alice.fr> (rubrique Développement).

N'hésitez pas, également, à venir laisser vos commentaires, poser vos questions, ou simplement proposer votre aide sur [www.wxdevelop.com](http://www.wxdevelop.com)

@+, et bonne prog. !