Installation de wxWidgets sur Windows

par 3DArchi (Ressources proposées par 3DArchi)

Date de publication : 20 mai 2009

Ce tutoriel décrit la procédure d'installation et de compilation de wxWidgets sur une plateforme Windows avec les compilateurs C++ MinGW (GCC) et Visual C++ Express 2008. Avec wxWidgets, vous pourrez alors développer des applications multiplate-formes en C++ offrant une I.H.M. riche et des fonctionnalités étendues : architecture M.V.C. (Modèle - Vue - Document), réseaux (FTP, HTTP, HTML, TCP, UDP, socket), XML, multi-threading, base de données (ODBC), OpenGL, etc. Plus besoin de modifier votre code pour générer vos applications sous Windows, Windows CE, Linux ou MAC. Cette bibliothèque gratuite et avec une licence souple offre une alternative aux MFC de Microsoft ou à Qt de Trolltech.
Donnez votre avis sur cet article :

I - Introduction

I-A - Révisions

I-A-1 - Document

Date	Révision article
Mai 2009	Rédaction

I-A-2 - Outils

Ce tutoriel a été rédigé avec : **wxWidgets-2.8.10**.
Le code a été compilé et testé avec **Microsoft Visual Studio 2008 (Version 9.0.21022.8 RTM) VC Express Edition**. Le produit est disponible sur le **site de Microsoft**.
Le code a été compilé et testé avec **MinGW version 4.3.3**. La dernière version disponible de MinGW se trouve sur le **site officiel**. Cependant, les versions de MinGW sont souvent en retard sur les versions disponibles de GCC. Le site **Twilight Dragon Media** propose des versions **non officielles** de MinGW avec des versions plus récentes de GCC. C'est la version 4.3.3 disponible sur ce site qui a été utilisée.

I-B - Avant-propos

wxWidgets offre un cadre de classes C++ (framework) pour créer des applications, en particulier avec une interface graphique évoluée, indépendamment de la plateforme (Windows, MAC, Linux). wxWidgets est distribué sous sa propre licence : wxWindows Library Licence, Version 3.1. Les termes de la licence reprennent ceux de la LGPL (GNU Library General Public Licence) modulo un certain nombre d'exceptions dans le but de l'assouplir. Notamment, le point 2 :

```
2. The exception is that you may use, copy, link, modify and distribute
 under your own terms, binary object code versions of works based
 on the Library.
```

qui donne explicitement le droit de publier vos applications produites avec la bibliothèque sous la licence que vous souhaitez ; c'est à dire commerciale, libre ou autre ! Bien sûr, cela n'est pas contaminant à d'autres bibliothèques sous licence LGPL utilisée concomitamment avec wxWidgets.
Une des originalités de wxWidgets est d'offrir par défaut le rendu standard de la plateforme sur laquelle elle est compilée. Les applications construites pour Windows présentent les mêmes contrôles que ceux développés avec l'API Windows standard (win32 ou MFC), les applications MAC ont une I.H.M. MAC, etc. Une option permet aussi de choisir une I.H.M. spécifique, dite UNIVERSAL, lorsque l'application doit suivre une politique d'I.H.M. qui lui est propre et indépendante de la plateforme.
Pour plus d'information, vous pouvez consulter :

- Les différents tutoriels wxWidgets de developpez.com : **Tutoriels**
- Les forums wxWidgets de developpez.com : **Forum wxWidgets**
- Le site officiel de wxWidgets : **wxWidgets**
- La page outils C++ recense différents compilateurs : **Outils et compilateurs C++**

I-C - Remerciements

Je remercie **Alp** pour ses encouragements, **dourouc05** pour ses courageuses relectures.

Enfin, d'une façon plus globale, je remercie les membres des forums de developpez.com qui par la qualité de leurs interventions m'ouvrent constamment de nouvelles pistes de réflexion sur ma pratique de développement.

I-D - Téléchargement

I-D-1 - Les sources

Les sources sont disponibles sur le **site de wxWidgets**.
Le site propose soit de tout télécharger (wxAll), soit de télécharger un installeur pour Windows (wxMSW). Pour la rédaction de ce tutoriel, nous sommes partis de wxAll.

I-D-2 - La documentation

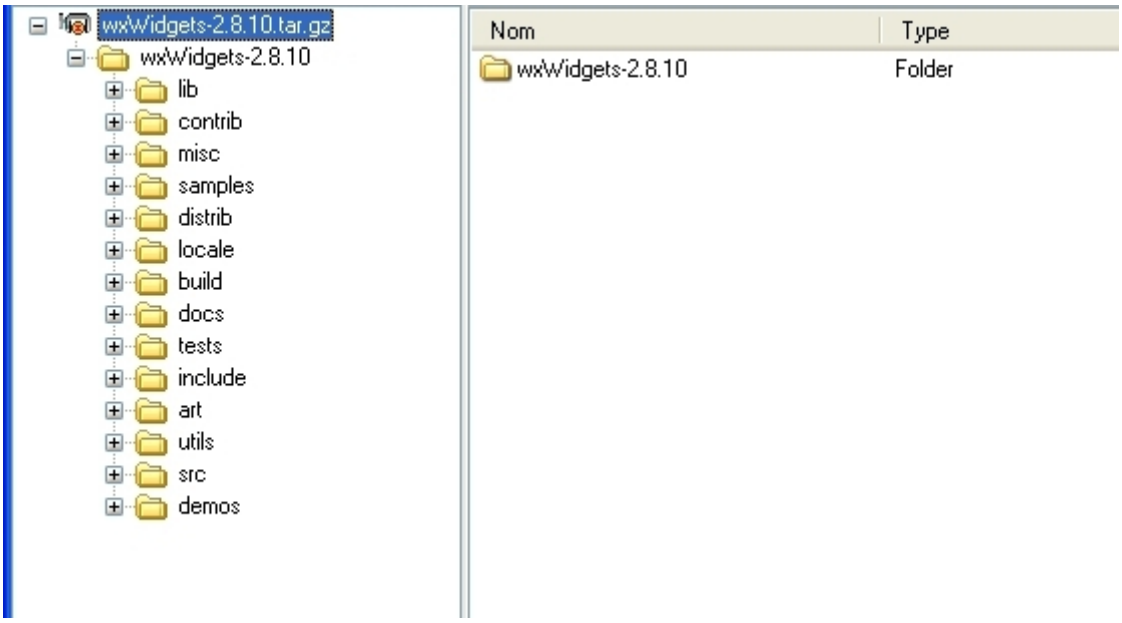
Afin de disposer de la documentation hors ligne, le site de wxWidgets propose les téléchargements suivants :

- HTML Docs : ensemble de fichiers HTML ;
- MS HTML Help Docs : ensemble de fichiers au format CHM (les fichiers d'aide de Windows) ;
- PDF Docs : ensemble de fichiers PDF ;
- WinHelp Docs : ensemble de fichiers au format HLP ;
- wxHTML Help Docs : ensemble de fichiers au format HTB.

Nous ne nous occuperons que des fichiers CHM. Les autres documentations suivent le même schéma.
La documentation reste disponible en ligne : **site wxWidgets en ligne de documentation**

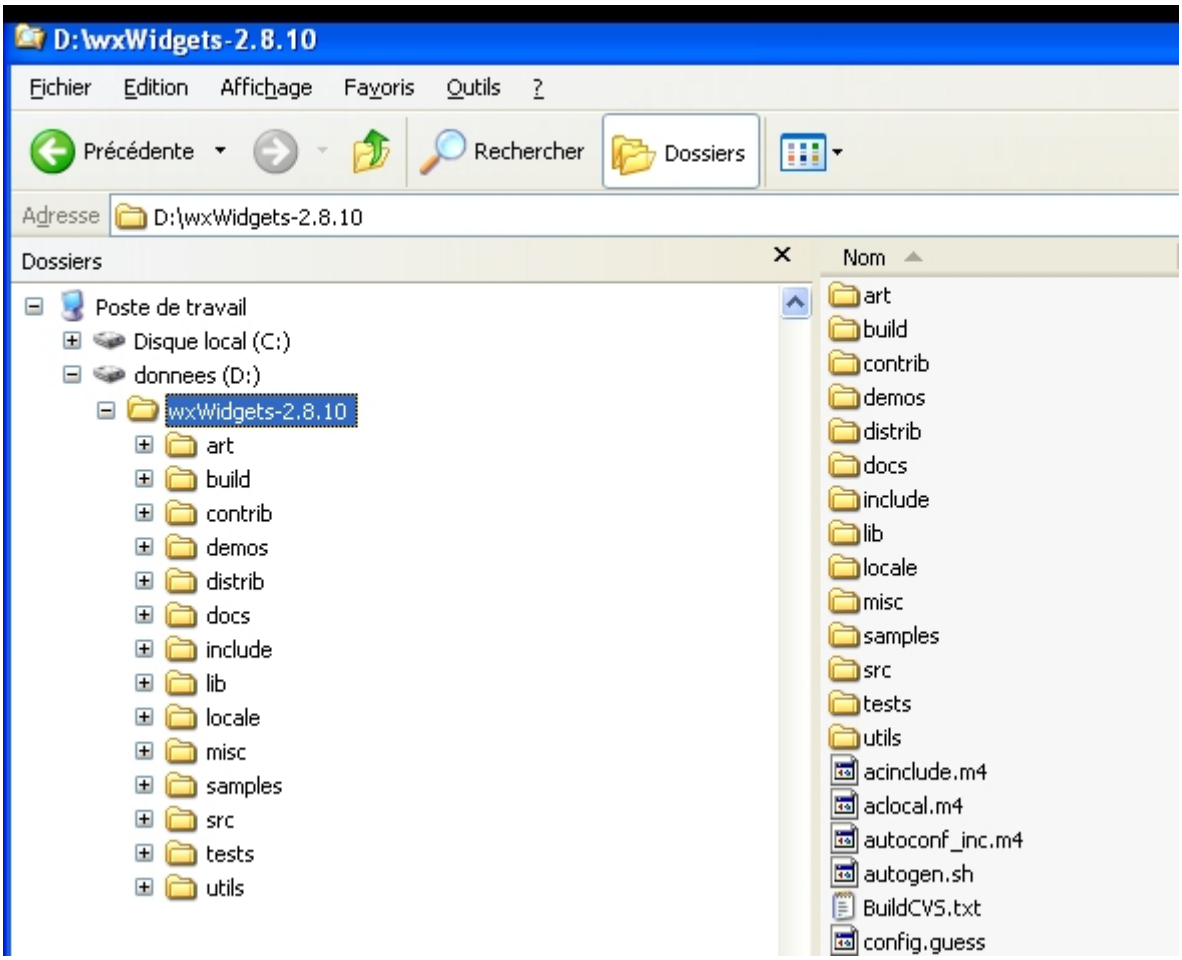
I-E - Extraire les fichiers

Pour commencer, je vous propose de décompresser le fichier contenant les sources vers le répertoire de votre choix. Le fichier compressé contient un répertoire wxWidgets-2.8.10 composé de l'ensemble des sous-répertoires de wxWidgets :



Structure du fichier compressé contenant les sources

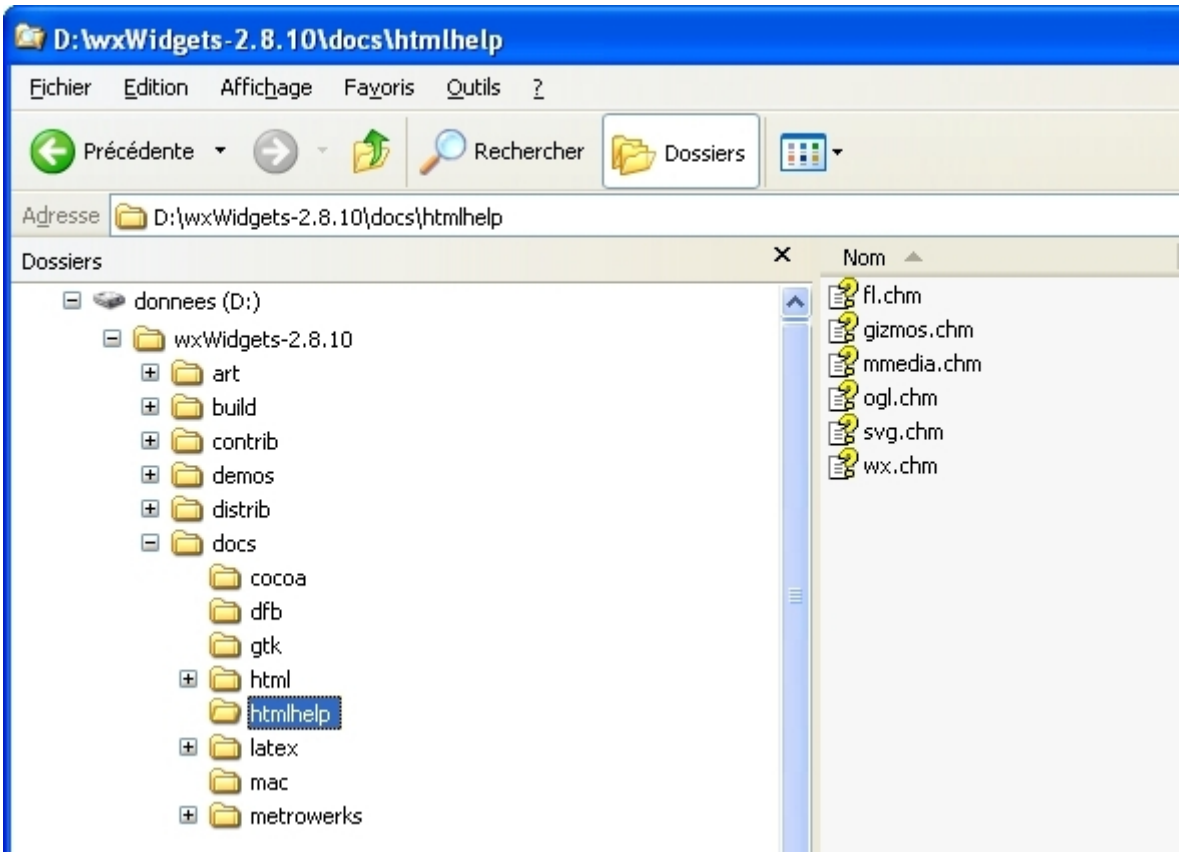
Décompressez ce fichier vers le répertoire de votre choix. Dans ce tutoriel, nous supposons qu'il s'agit de "D:", nous obtenons donc la hiérarchie suivante :



Hiérarchie des répertoires sources

La documentation précise bien que le chemin d'installation ne doit pas contenir d'espaces !

Ensuite, nous désarchivons la documentation. En choisissant le même répertoire que pour les sources, la documentation est directement insérée dans le répertoire adéquat : "\wxWidgets-2.8.10\docs". Ainsi, en désarchivant MS HTML Help Docs, le répertoire "\wxWidgets-2.8.10\docs\htmlhelp" est créé. Il contient les fichiers d'aide .chm.



Les fichiers d'aide

I-F - Les différentes compilations

On peut distinguer différentes compilations orthogonales entre elles :

- DEBUG ou RELEASE : pour opter entre la version debug ou release.
- STATIC ou SHARED : pour obtenir des bibliothèques statiques ou DLL. La version statique permet de résoudre les liens à la compilation, pendant la phase d'édition des liens. Votre exécutable peut alors être distribué tel quel sans DLL supplémentaire. La contrepartie est un exécutable de taille conséquente. En mode partagé (SHARED), les DLLs sont générées avec le .lib associé. L'édition de lien est alors partielle et vous devez distribuer votre exécutable avec les DLLs.
- UNICODE ou non : pour générer une version UNICODE ou non de la bibliothèque.
- UNIVERSAL ou MSW : permet de choisir entre un rendu propre à wxWidgets ou le rendu de la plateforme (Windows).
- MONOLITHIC : génère une seule et unique bibliothèque ou plusieurs bibliothèques par sujet (I.H.M., XML, réseau, etc.)

Cela fait plus d'une trentaine de générations possibles. Il est clair que toutes ne sont pas nécessaires dans un projet. En général, les versions DEBUG et RELEASE sont générées. Les autres options sont fixées dès le début pour le projet voire pour tous les projets.
Dans le cadre de ce tutoriel, nous allons générer les versions DEBUG et RELEASE des bibliothèques STATIC avec UNICODE, le port MSW et en plusieurs bibliothèques.

Le répertoire "\wxWidgets-2.8.10\build\msw" contient les fichiers nécessaires à la génération :

- les fichiers de configuration ;
- les makefile ;
- pour Visual C++, les fichiers projets.

Chacun de ces fichiers est lié au compilateur utilisé pour la génération. Nous allons donc voir ci-dessous comment procéder.

II - wxWidgets et MinGW

II-A - Paramétrer la compilation

Pour MinGW, le fichier de configuration est "\wxWidgets-2.8.10\build\msw\config.gcc" et le makefile est "\wxWidgets-2.8.10\build\msw\makefile.gcc". Inutile de modifier le second. Seul le premier - "\wxWidgets-2.8.10\build\msw\config.gcc" - a pour objectif d'être paramétré. Les premières lignes de ce fichier concernent les différents paramètres de compilation sur lesquels nous n'intervenons pas. Nous allons commencer à nous intéresser aux directives de préprocessing commençant avec SHARED ?= 0. Toutes celles qui suivent ont la même forme. Le tableau suivant les récapitule en précisant leur objectif, la valeur par défaut et la valeur que nous allons choisir :

Directive	Défaut	Nouvelle	Objectif
SHARED	0	0	Une valeur de 1 permet de générer les DLL. Avec une valeur de 0, nous allons générer uniquement des .lib liés statiquement à la compilation.
WXUNIV	0	0	Opte entre un rendu natif de Windows (0) ou le rendu UNIVERSAL de wxWidgets (1).
UNICODE	0	1	Nous avons choisi de générer la version UNICODE donc nous positionnons la valeur à 1.
MSLU	0	0	Cette directive concerne la génération des bibliothèques UNICODE pour Windows 9x. Elle doit être positionnée à 1 en conjonction avec la valeur précédente pour cette plateforme.
BUILD	debug	debug	Peu importe la valeur positionnée ici car nous spécifierons la génération DEBUG ou RELEASE lorsque nous demanderons la compilation en ligne de commande. La valeur est donc ignorée. A noter que les valeurs textuelles, celle-ci comme les suivantes, sont sensibles à la casse.
DEBUG_INFO	default	default	Positionnées à *default*, les informations de DEBUG sont en cohérence avec la directive précédente : si BUILD vaut 'debug', les informations sont insérées. Et quand BUILD prend la valeur 'release', les informations ne sont pas ajoutées. En positionnant DEBUG_INFO à 1 ou à 0, les informations de DEBUG ne dépendent plus du type de génération mais sont systématiquement (valeur 1) ou jamais (valeur 0) ajoutées. En fait, ce n'est pas tant la valeur 0 qui est intéressante, mais la possibilité avec la valeur 1 d'avoir des informations de debug y compris en mode release. Pour notre génération, nous conservons la valeur 'default' pour n'avoir les informations de DEBUG que pour la version DEBUG.
DEBUG_FLAG	default	default	Cette directive ressemble en tout point à la précédente si ce n'est qu'elle sert à positionner la directive de compilation __WXDEBUG__ ; c'est cette dernière qui est utilisée dans le code

			wxWidgets pour séparer le code généré en version DEBUG ou RELEASE.
MONOLITHIC	0	0	Avec une valeur à 0, plusieurs bibliothèques (lib et/ou DLL) sont générées permettant de construire une application avec les seuls composants utilisés. Avec une valeur de 1, la compilation ne produit qu'une seule bibliothèque embarquant tous les composants.
USE_GUI	1	1	Si vous souhaitez construire des applications sans I.H.M., la valeur peut être positionnée à 0. Nous la maintenons à 1 car nous voulons construire des applications avec l'API I.H.M. de wxWidgets.
USE_HTML	1	1	Contrôle la production de wxHTML.lib.
USE_MEDIA	1	1	Concerne la bibliothèque wxMedia proposant des services multimédias.
USE_XRC	1	1	wxWidgets offre un système de ressources (boîte de dialogue, menu, icônes, etc.) basé sur un format XRC. En activant cette possibilité, les applications peuvent déporter les spécifications de leur I.H.M. dans ces fichiers ressources séparés.
USE_AUI	1	1	Produit la bibliothèque wxAUI (Advanced User Interface) : composant pour la définition d'interface avancée.
USE_RICHTEXT	1	1	Pilote la production de la bibliothèque wxRichText.
USE_OPENGL	0	1	Afin d'utiliser les composants wxWidgets avec OpenGL, la variable prend la valeur 1.
USE_ODBC	0	1	Afin d'activer la génération de la bibliothèque de liaison ODBC, nous modifions la valeur à 1.
USE_QA	0	0	Pilote la création de la bibliothèque d'assurance qualité.
USE_EXCEPTIONS	1	1	Positionné à 1 permet à wxWidgets d'être 'exception safe'. Avec une valeur de 0, le code produit est plus rapide et plus petit, mais la génération d'une exception aboutit à un comportement indéterminé. Le mécanisme d'exception étant à la base du développement d'applications robustes en C++, je ne peux que conseiller de maintenir cette option à 1.
USE_RTTI	1	1	Gouverne l'utilisation du RTTI (Run Time Type Information) c'est-à-dire l'information de type à l'exécution.
USE_THREADS	1	1	Quelle application aujourd'hui ne cache pas un petit thread. Ne nous privons donc pas des mécanismes wxWidgets de gestion du multithreading.
USE_GDIPLUS	0	0	Permet l'utilisation de GDI+. Il semblerait que l'utilisation de GDI+ avec MinGW soit laborieuse et la plus part des ressources recommandent de ne pas générer wxWidgets avec GDI+ en utilisant MinGW.
GCC_VERSION	3	3	Tout simplement 3 pour toute version supérieure à ou égale à 3. Historiquement, il semblerait que wxWidgets avait quelques problèmes à compiler avec les versions antérieures de MinGW/GCC.

Une fois le fichier "\wxWidgets-2.8.10\build\msw\config.gcc" modifié, nous devons également intervenir sur le fichier "\wxWidgets-2.8.10\include\wx\msw\setup.h". La différence ? "\wxWidgets-2.8.10\build\msw\config.gcc" gouverne la compilation des bibliothèques wxWidgets. "\wxWidgets-2.8.10\include\wx\msw\setup.h" est l'en-tête reprenant cette configuration dans vos projets. Il faut donc ajuster quelques définitions de directives pour maintenir la cohérence. En l'occurrence, nous allons modifier les directives suivantes :

- #define wxUSE_UNICODE 1 : car nous voulons compiler avec UNICODE
- #define wxUSE_GLCANVAS 1 : car nous voulons compiler avec USE_OPENGL
- #define wxUSE_ODBC 1 : car nous voulons compiler avec USE_ODBC

A cela, nous modifions les directives suivantes :

- #define wxUSE_STL 1 : wxWidgets définit ses propres classes pour les listes (wxList) et les tableaux (wxArray). En positionnant cette directive à 1, ces deux classes vont dériver de std::list et std::vector tirant ainsi profit de la STL.
- #define wxUSE_STD_IOSTREAM 1 : grâce à cette directive, nous indiquons que nous souhaitons utiliser les flux de la bibliothèque standard quand cela est possible à la place des flux définis dans wxWidgets.

II-B - Compiler

Pour compiler, nous supposons que la variable d'environnement PATH contient le chemin vers MinGW/bin. C'est dans ce répertoire que se trouvent le compilateur, l'éditeur de lien, le make, etc.
La compilation de wxWidgets en mode DEBUG s'écrit simplement :

```
mingw32-make -f makefile.gcc BUILD=debug
```

Attention à bien respecter la casse : BUILD en majuscule et debug en minuscule. Sans quoi, la version générée est celle donnée par la valeur de la variable BUILD du fichier de configuration !

La compilation peut provoquer quelques avertissements. Il ne faut pas en tenir compte.
Les fichiers intermédiaires de la compilation (les .o) sont générés dans "wxWidgets-2.8.10\build\msw\gcc_mswud". Le nom du répertoire intermédiaire est composé à partir de :

- le compilateur : 'gcc' ;
- la plateforme : 'msw' ;
- le mode DEBUG/RELEASE : 'd' pour debug, rien pour release ;
- le mode UNICODE : 'u' pour UNICODE.

Si nous avions choisi le mode non UNICODE, le répertoire aurait eu la forme "gcc_mswd". Et comme nous allons le voir, la compilation en mode RELEASE crée le répertoire "gcc_mswu".

La compilation ajoute un autre répertoire : "\wxWidgets-2.8.10\lib\gcc_lib". Le schéma suivi pour le nom du répertoire ne prend en compte que le nom du compilateur utilisé. Toutes les versions des bibliothèques compilées avec un même compilateur (DEBUG, RELEASE, UNICODE, ANSI) sont mises dans ce même répertoire ce qui facilite le paramétrage de l'I.D.E. (ou du makefile si vous préférez mettre les mains dans le cambouis). Ce répertoire contient à sa base l'ensemble des bibliothèques générées :

- libwxbase28ud.a
- libwxbase28ud_net.a
- libwxbase28ud_odbc.a
- libwxbase28ud_xml.a
- libwxexpatd.a
- libwxjpegd.a
- libwxmsw28ud_adv.a

- libwxmsw28ud_aui.a
- libwxmsw28ud_core.a
- libwxmsw28ud_dbgrid.a
- libwxmsw28ud_gl.a
- libwxmsw28ud_html.a
- libwxmsw28ud_media.a
- libwxmsw28ud_qa.a
- libwxmsw28ud_richtext.a
- libwxmsw28ud_xrc.a
- libwxpngd.a
- libwxregexud.a
- libwxtiffd.a
- libwxzlibd.a

Chaque bibliothèque porte un nom suivant le schéma :

```
lib[nom]ud.a
```

Certaines n'ont pas le suffixe 'u' dès lors qu'elles ne dépendent pas du critère UNICODE.
Toujours dans "\wxWidgets-2.8.10\lib\gcc_lib" un répertoire a été inséré avec les bibliothèques : "\wxWidgets-2.8.10\lib\gcc_lib\mswud". Celui-ci contient un premier fichier : "build.cfg". Ce fichier récapitule les options positionnées dans config.gcc. Cela permet de se souvenir quelle a été la configuration utilisée pour générer la version. A côté de ce fichier, un dernier répertoire "\wxWidgets-2.8.10\lib\gcc_lib\mswud\wx" contient un fichier "setup.h". Il s'agit du fichier d'en-tête correspondant à la compilation. Nous verrons plus loin qu'il est important de bien configurer son I.D.E. de façon à ce que chaque ligne *#include <wx/setup.h>* corresponde à l'inclusion de ce fichier d'en-tête et non pas à celui dans "\wxWidgets-2.8.10\include\wx\msw\setup.h".

De façon identique, la compilation en RELEASE s'écrit :

```
mingw32-make -f makefile.gcc BUILD=release
```

Les fichiers intermédiaires sont créés dans le répertoire "wxWidgets-2.8.10\build\msw\gcc_mswu".
Les bibliothèques générées dans "\wxWidgets-2.8.10\lib\gcc_lib" sont les mêmes que précédemment mais sans le suffixe 'd' pour DEBUG :

- libwxbase28u.a
- libwxbase28u_net.a
- libwxbase28u_odbc.a
- libwxbase28u_xml.a
- libwxexpat.a
- libwxjpeg.a
- libwxmsw28u_adv.a
- libwxmsw28u_aui.a
- libwxmsw28u_core.a
- libwxmsw28u_dbgrid.a
- libwxmsw28u_gl.a
- libwxmsw28u_html.a
- libwxmsw28u_media.a
- libwxmsw28u_qa.a
- libwxmsw28u_richtext.a
- libwxmsw28u_xrc.a
- libwxpng.a
- libwxregexu.a
- libwxtiff.a
- libwxzlib.a

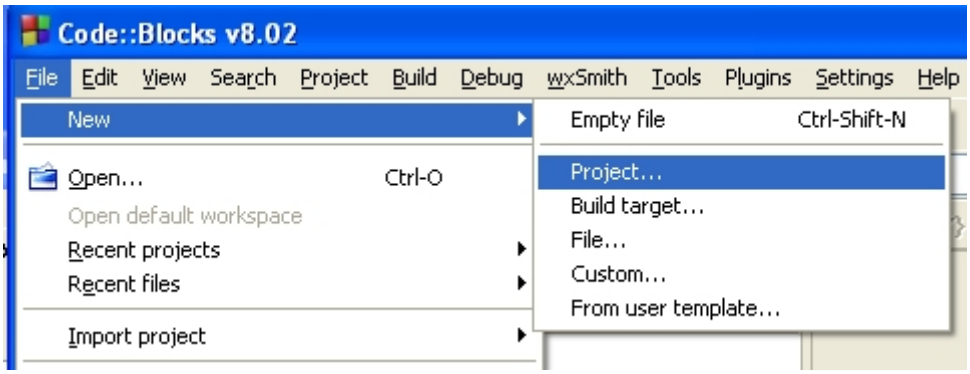
Le répertoire "\wxWidgets-2.8.10\lib\gcc_lib\mswu" contient le fichier "build.cfg" et le répertoire "wx" avec son fichier "setup.h".

II-C - Paramétrer son I.D.E.

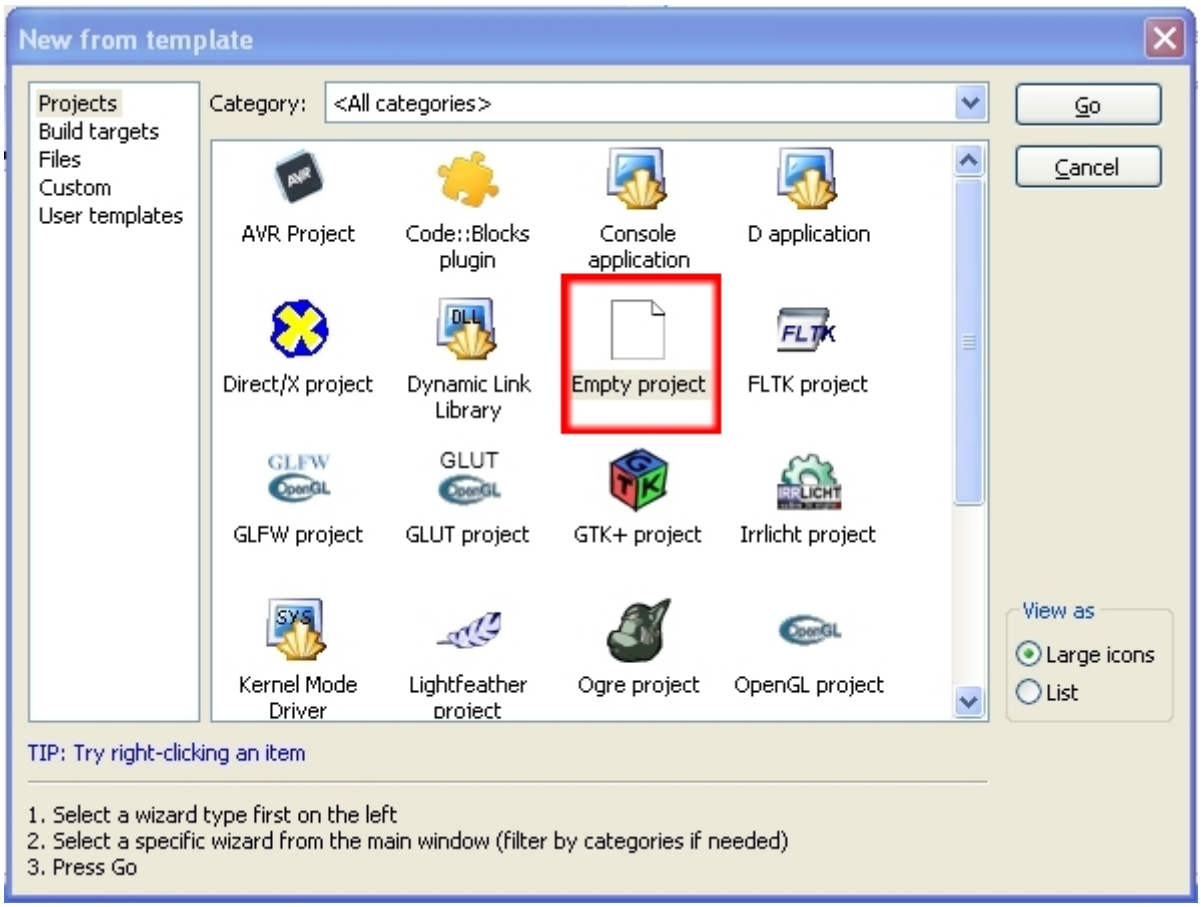
Pour MinGW nous proposons la configuration de Code::Block.

II-C-1 - A partir d'un projet existant

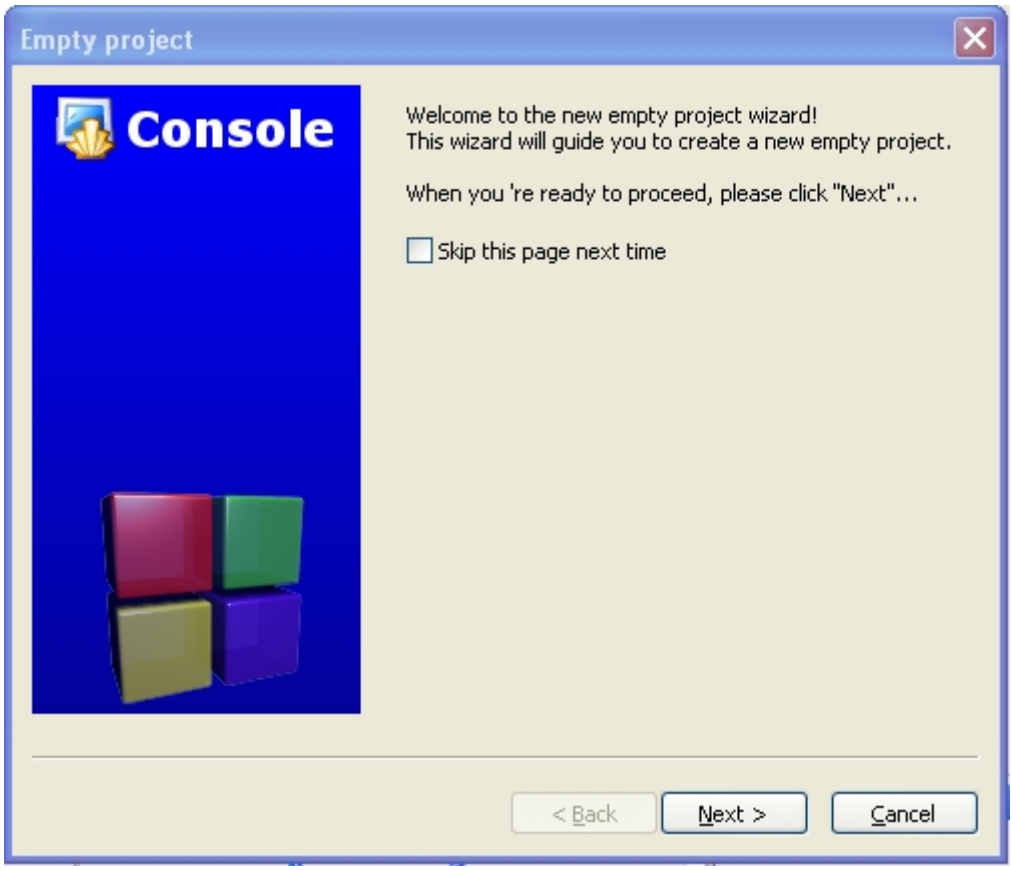
Pour montrer la configuration pour un projet existant, nous allons commencer par créer ce projet. Sous Code::Block, nous choisissons le menu 'File/New/Project' :



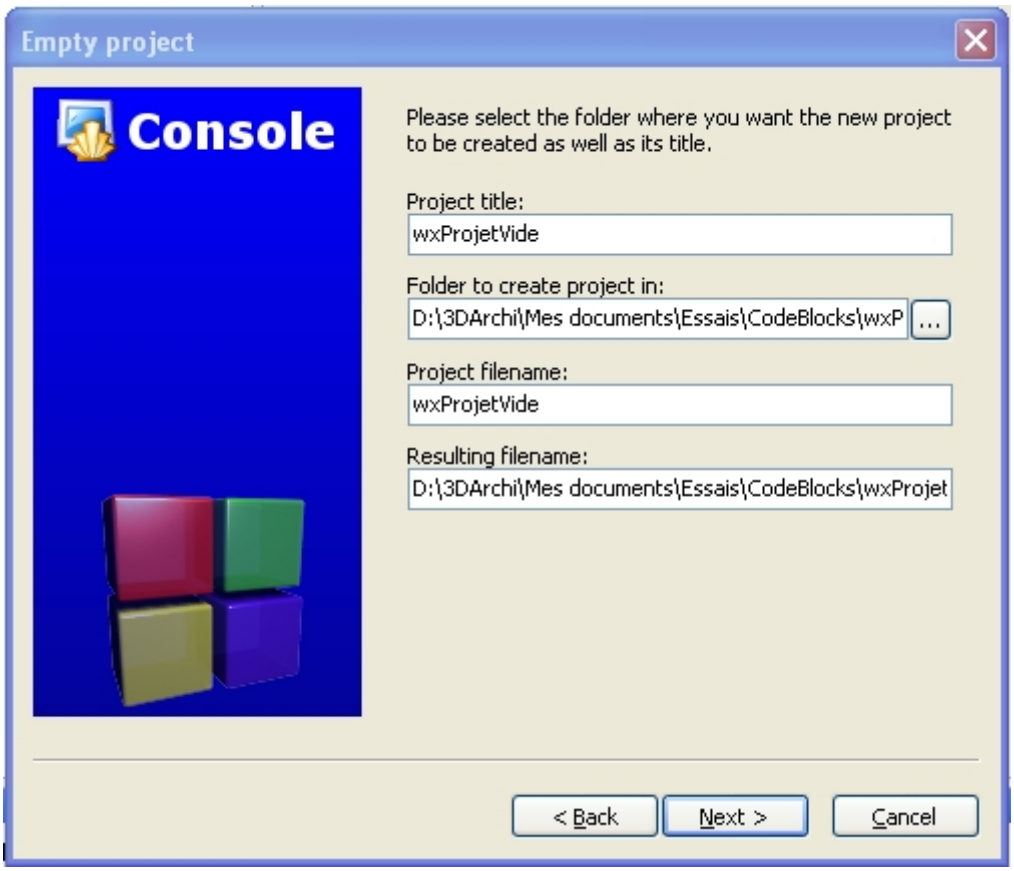
Nous choisissons un projet vide :



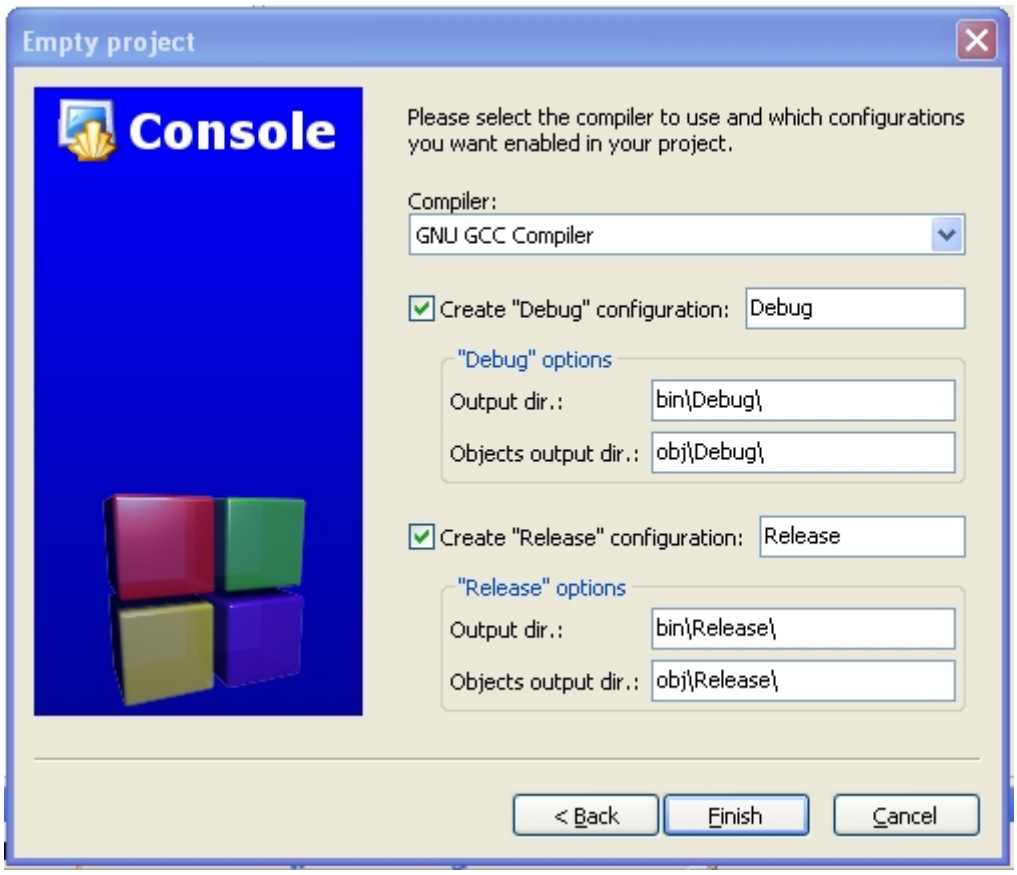
Apparaît alors un assistant pour la création du projet :



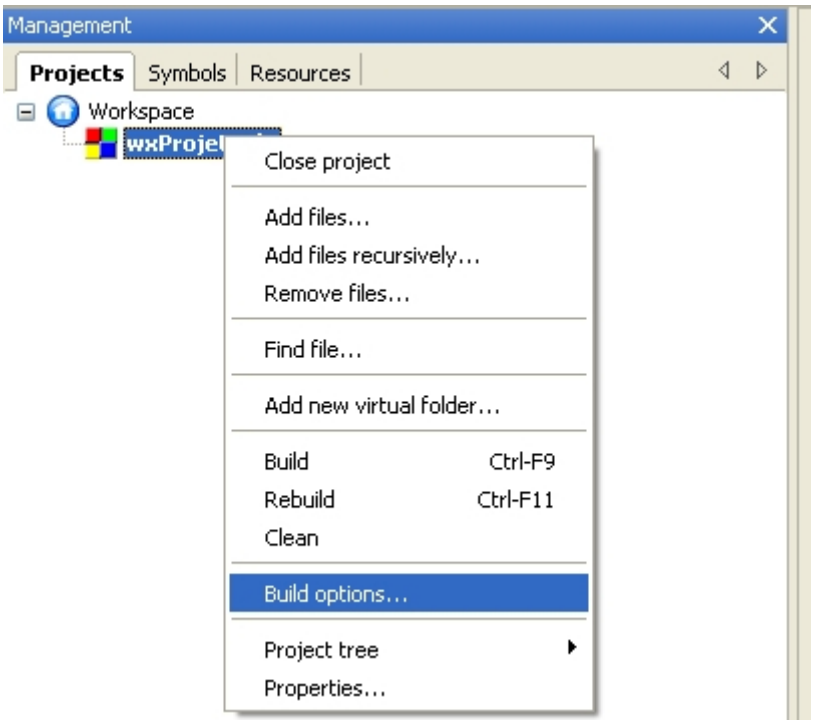
Nous choisissons suivant et obtenons la page de saisie du titre du projet et du répertoire de création, le nom du projet et le fichier correspondant sont automatiquement construits avec ces deux informations :



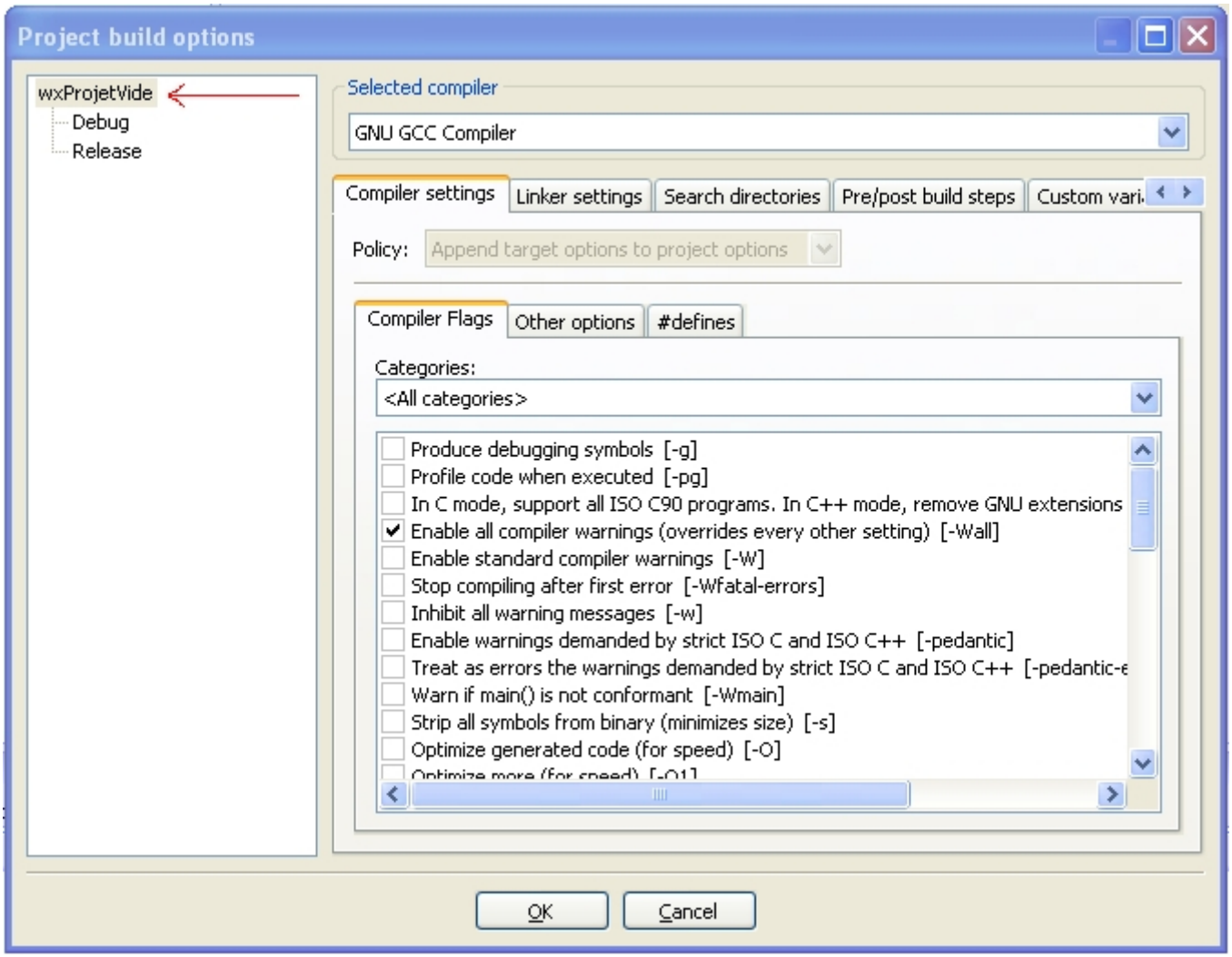
La page suivante nous propose de choisir le compilateur : GNU GCC Compiler pour la compilation avec MinGW (nous supposons que vous avez correctement paramétré Code::Block pour aller chercher le compilateur MinGW correspondant).
Nous gardons cochées les cases pour la configuration Debug et Release :



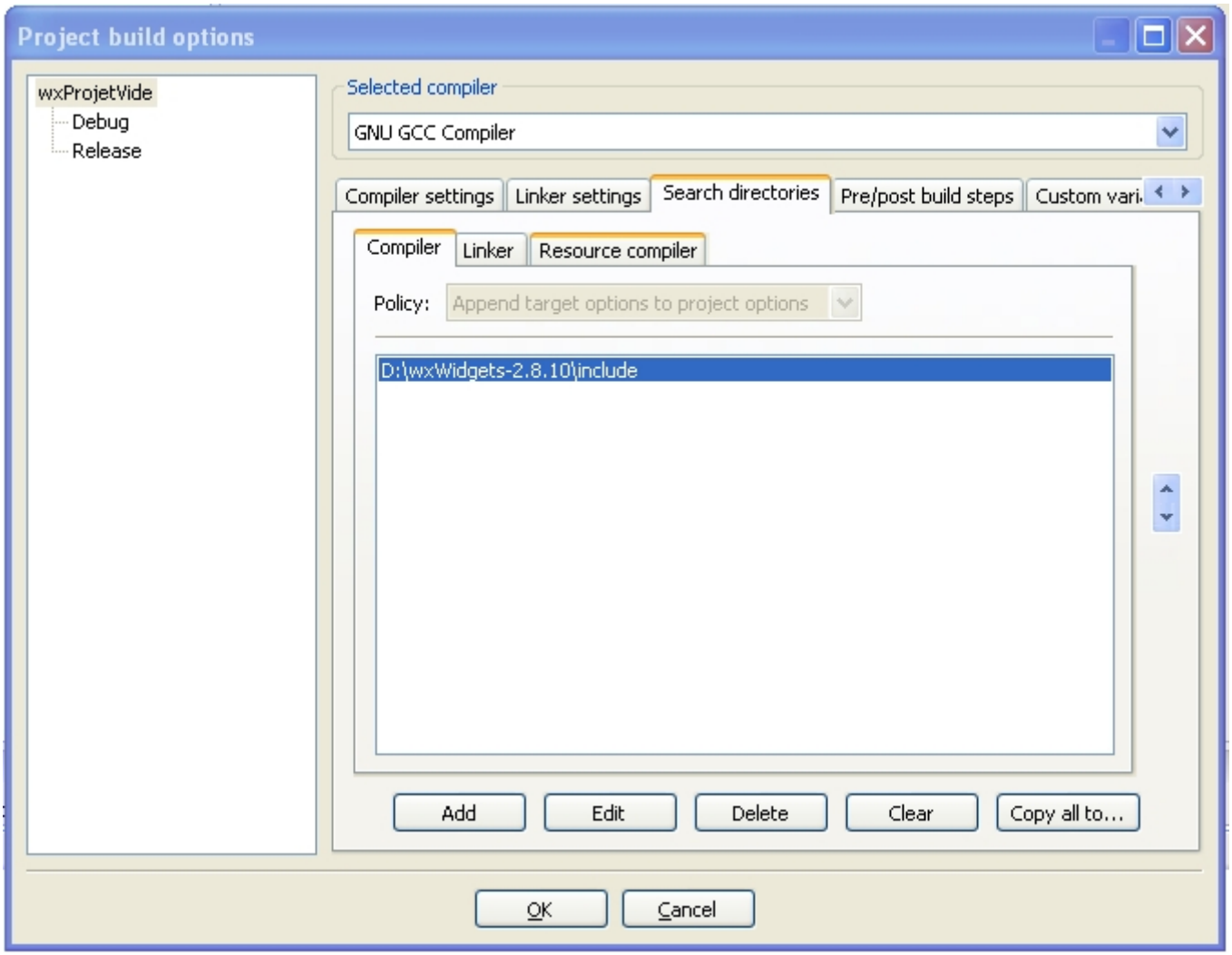
Nous voilà maintenant avec notre projet vide. Nous allons le configurer pour l'utilisation de wxWidgets.
Dans le menu contextuel de notre nouveau projet, nous choisissons l'item 'Build Options...'



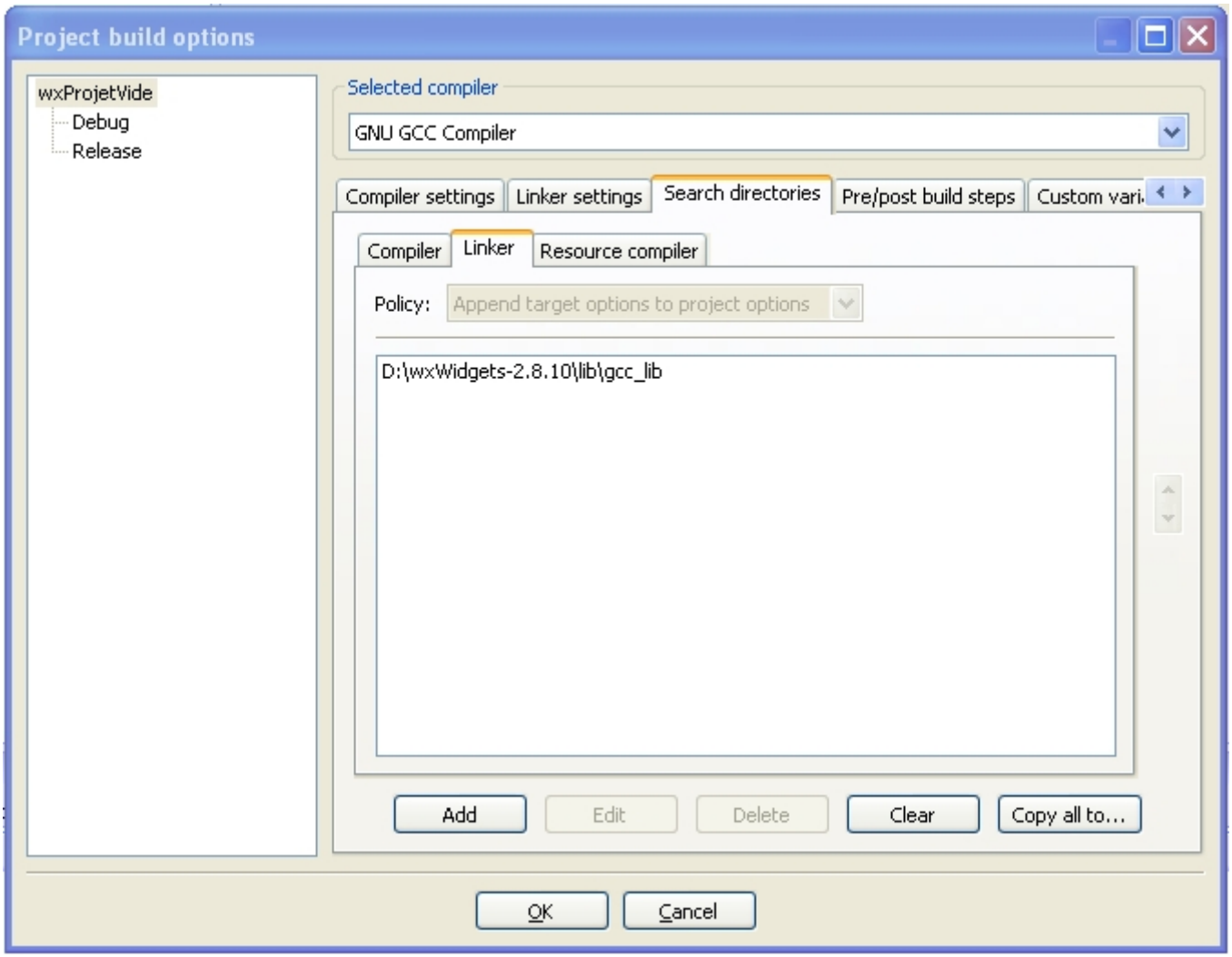
Nous commençons par positionner les paramètres pour toutes les configurations de génération. Il faut donc choisir le nom du projet en haut de l'arbre de gauche :



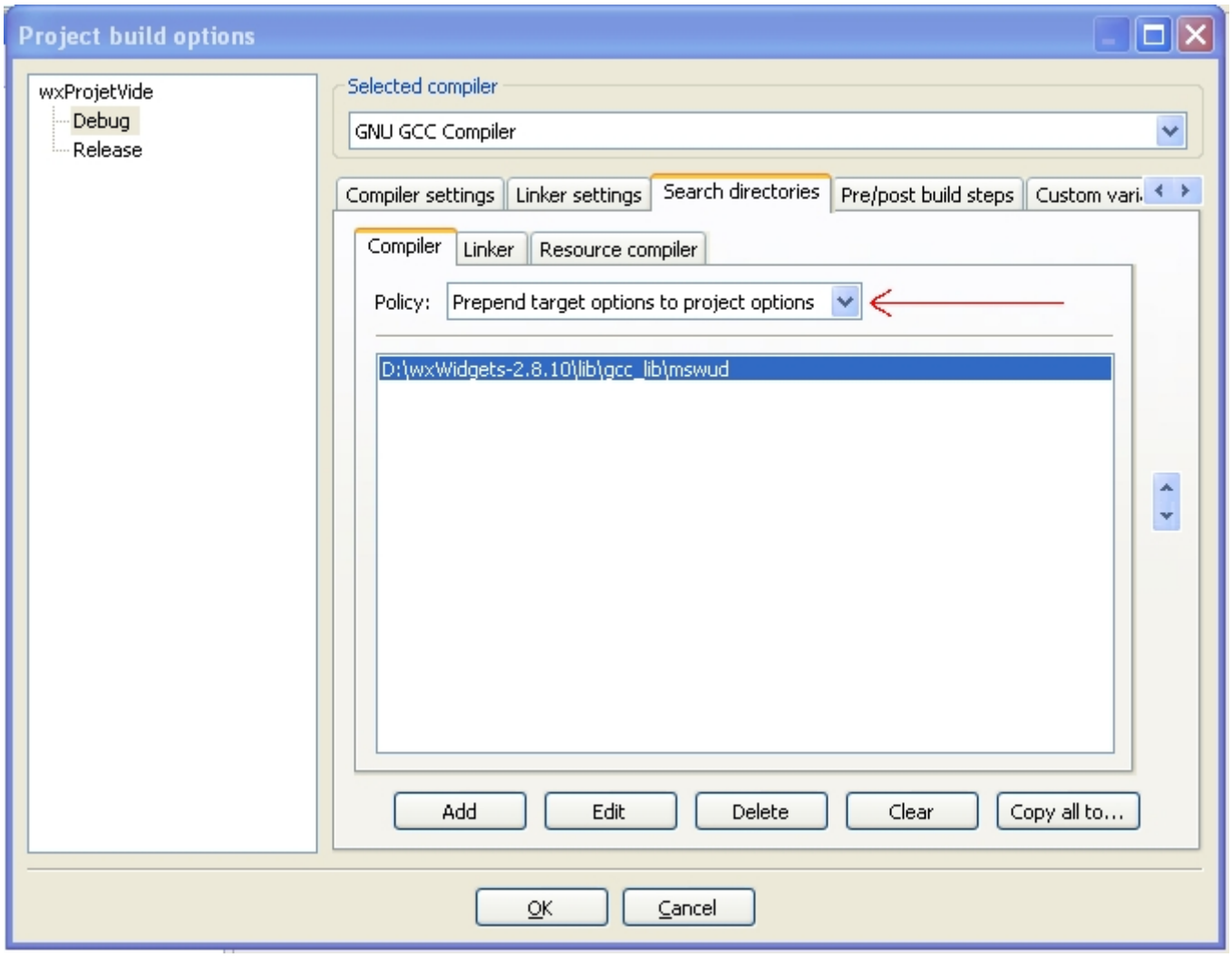
Nous choisissons l'onglet 'Search directories' et le sous- onglet 'Compiler'. Nous ajoutons "\wxWidgets-2.8.10\include". Code::Block demande si le chemin doit être ajouté en relatif. Pour une bibliothèque, je préconise de choisir Non pour conserver le chemin en absolu :



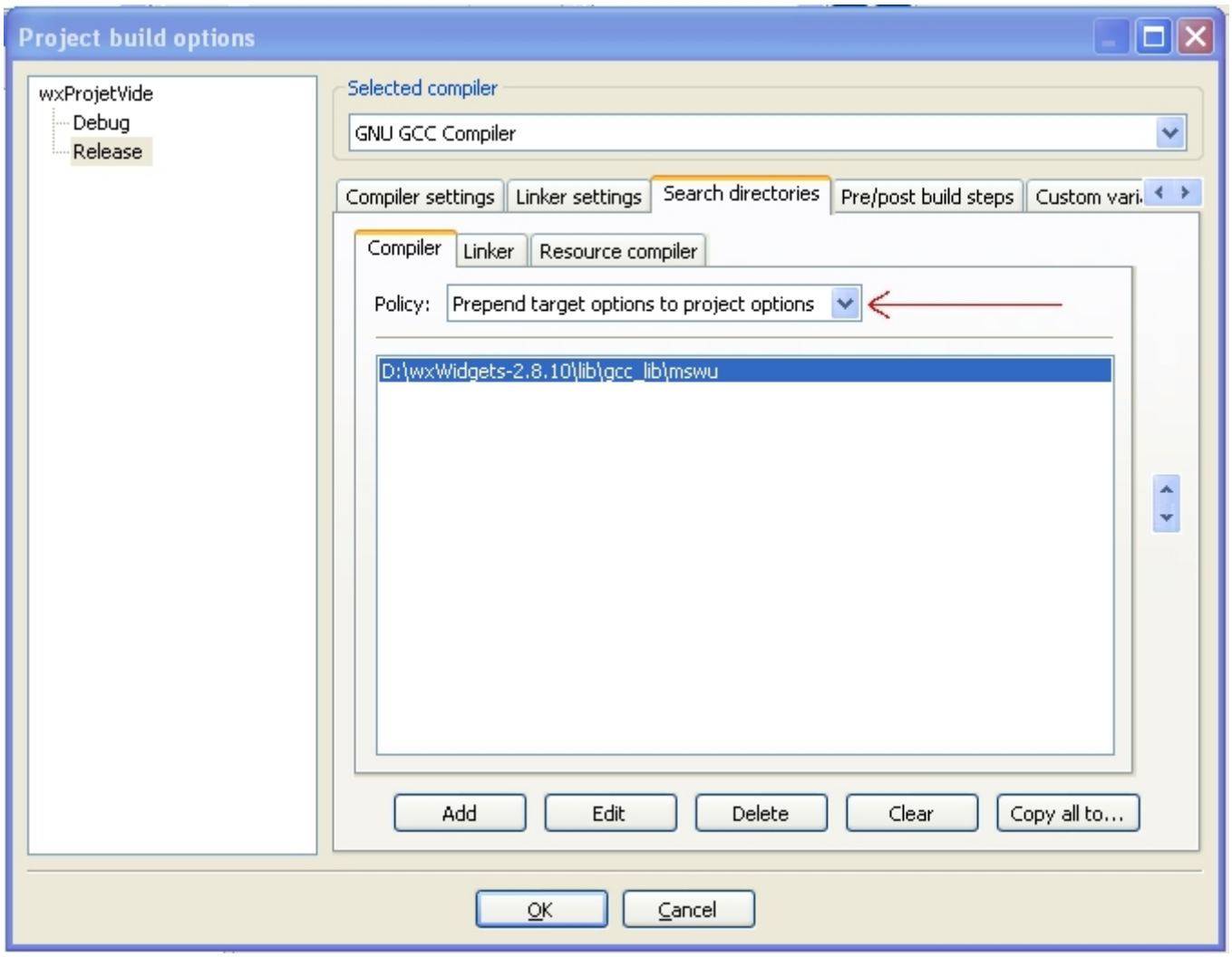
Le même paramétrage doit être fait pour le chemin d'accès aux bibliothèques .lib précédemment générées. Cela se fait dans le sous-onglet Linker :



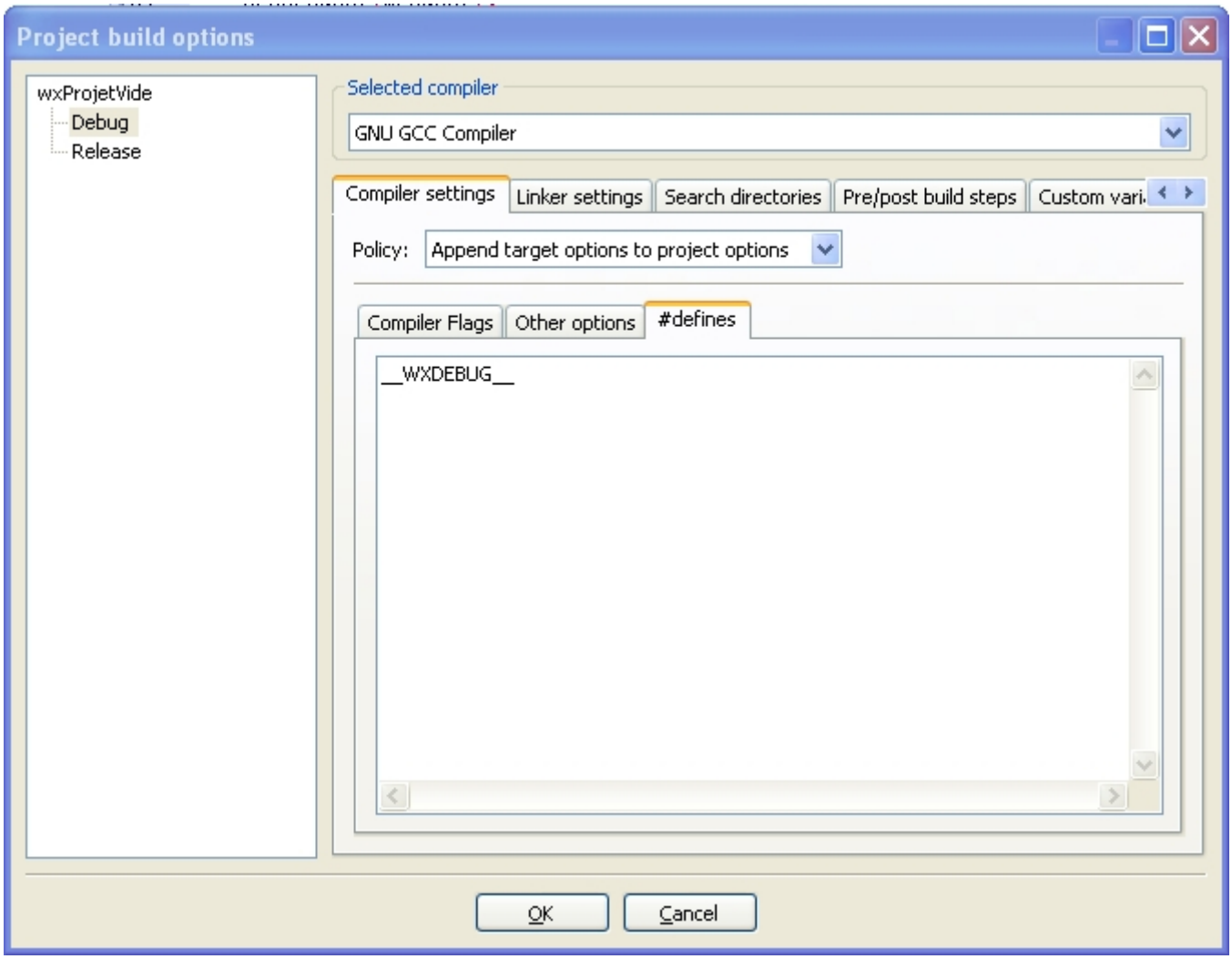
Maintenant, nous configurons le mode debug. Dans l'onglet de gauche, nous sélectionnons 'Debug' et dans la page de droite nous sélectionnons l'onglet 'Search directories' et le sous-onglet 'Compiler'. Nous ajoutons "\wxWidgets-2.8.10\lib\gcc_lib\mswud" et dans la boîte déroulante nous choisissons 'Prepend target options to project options'. De cette façon les fichiers (et setup.h) sont d'abord cherchés dans ce répertoire puis dans le répertoire précisé pour tout le projet :



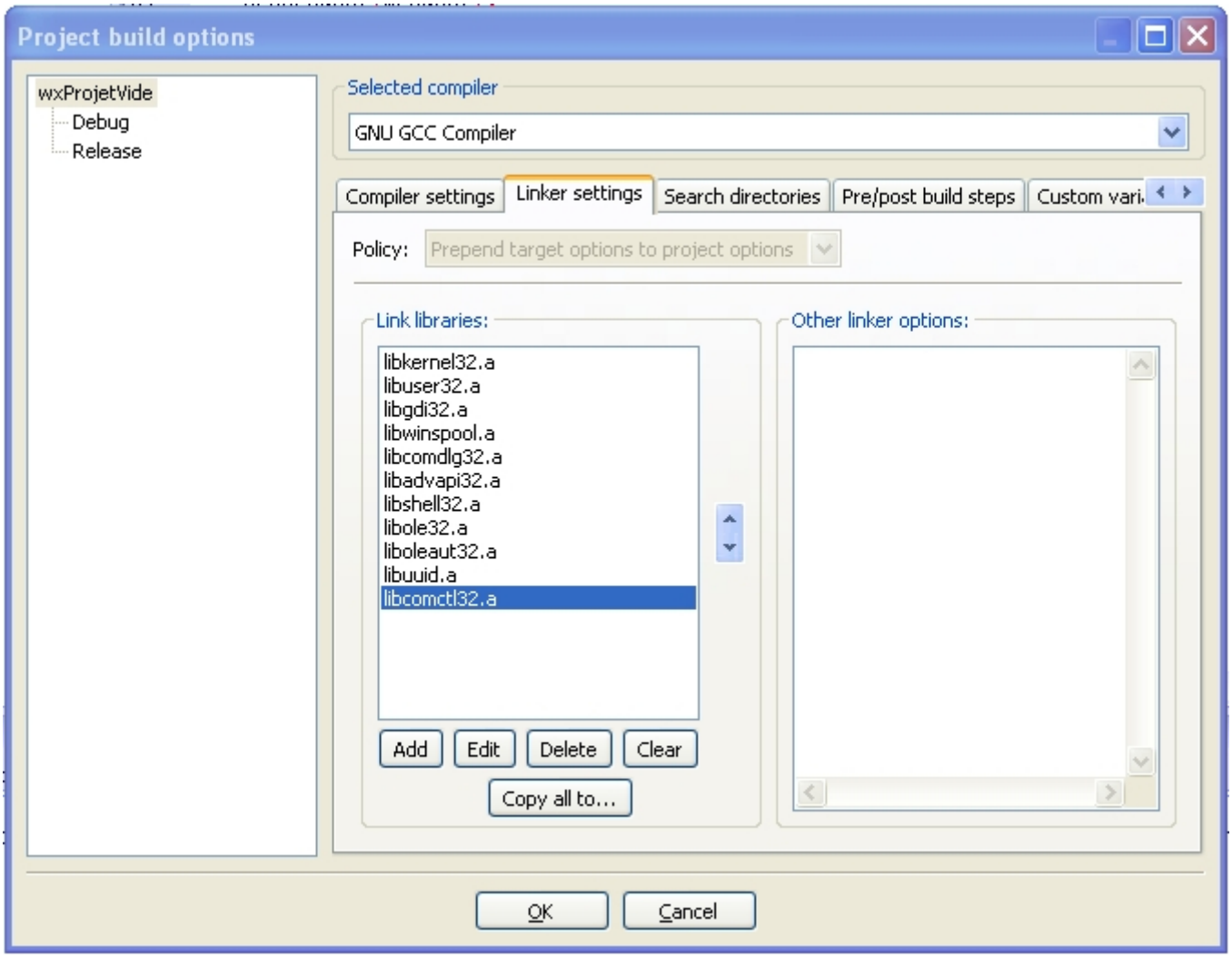
L'opération est renouvelée pour la configuration Release avec le répertoire "\wxWidgets-2.8.10\lib\gcc_lib\mswu"



Nous positionnons l'option de compilation __WXDEBUG__ pour la version debug dans l'onglet 'Compiler settings' et le sous-onglet '#define' :

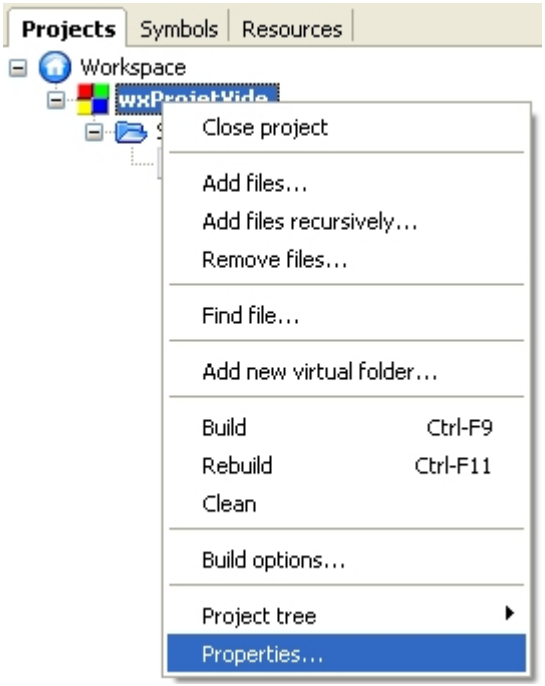


Et comme nous avons créé un projet vide, nous devons aussi spécifier les bibliothèques standards de Windows dans l'onglet 'Linker settings' pour l'ensemble des générations :

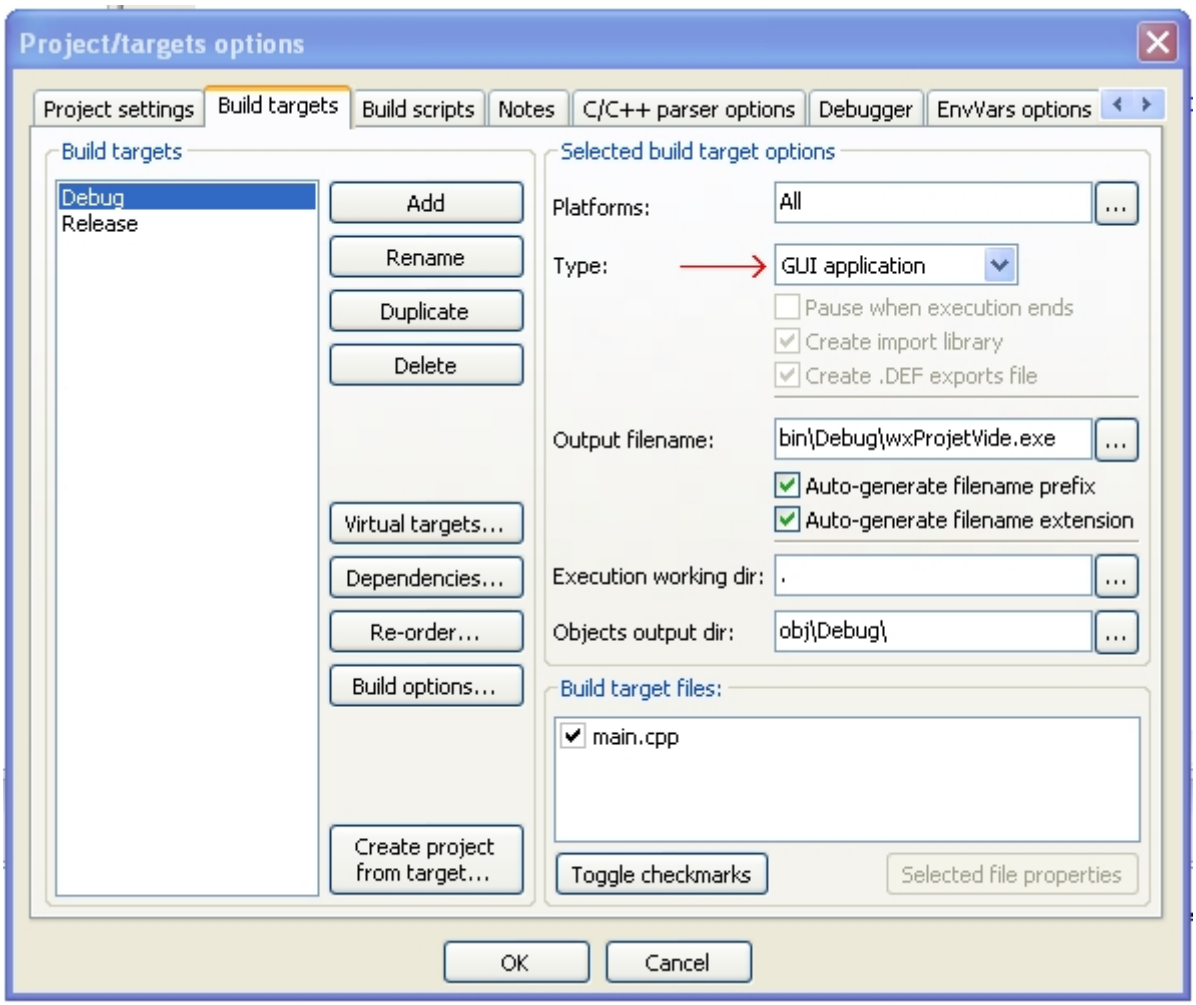


L'ordre d'inclusion est important. Il faut conserver celui proposé sans quoi des erreurs à l'édition de lien peuvent apparaître.

Nous en avons provisoirement fini pour les options. Il nous reste un dernier réglage sur le projet. Nous choisissons donc l'item 'Properties' du menu contextuel de notre projet :



Dans l'onglet 'Build targets', dans la boîte déroulante 'Type', nous choisissions 'GUI application'. Sans quoi, notre application est lancée à partir d'une console. Et, bien sûr, la même option est positionnée pour la version release :



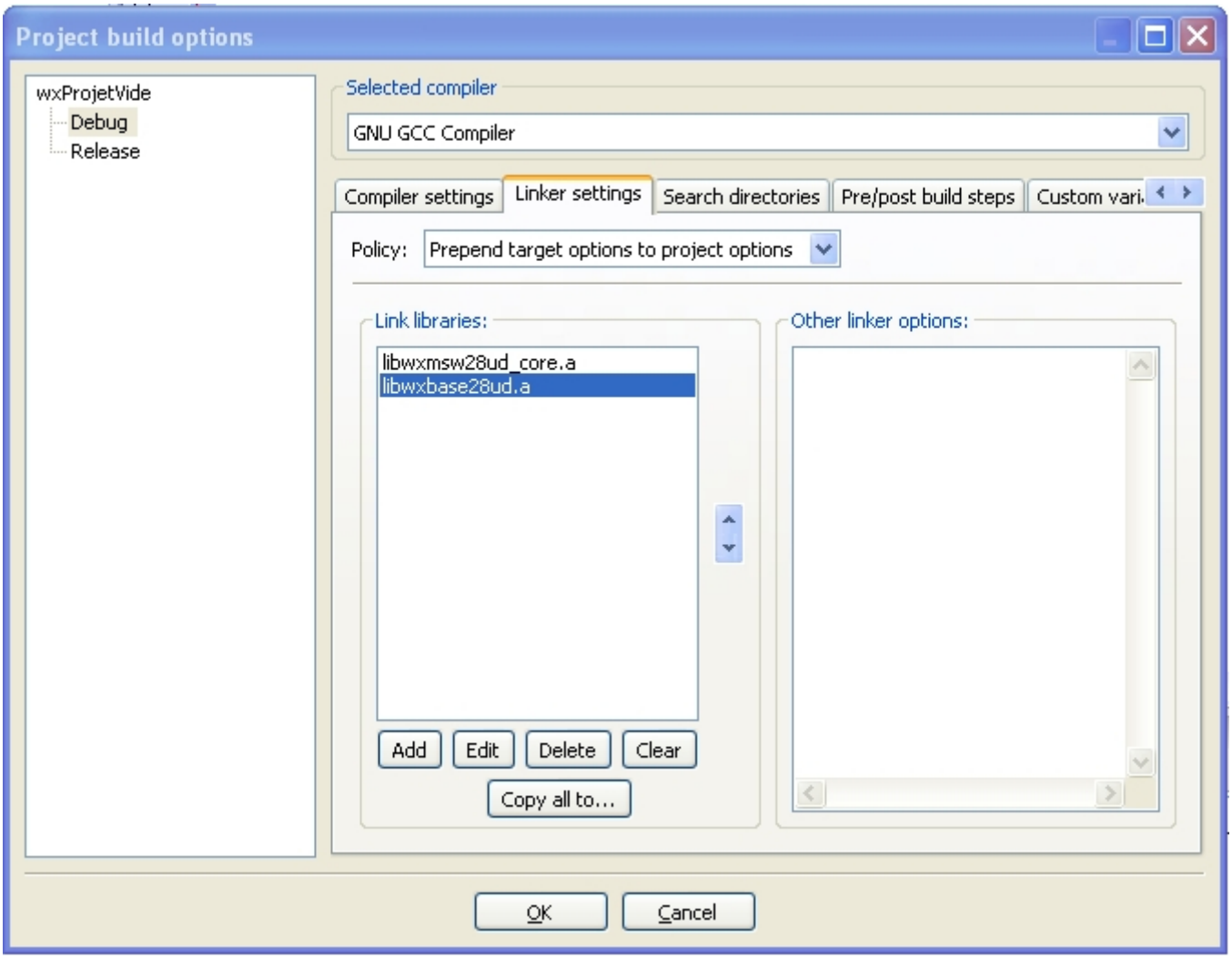
Nous créons un fichier vide "main.cpp" pour une application minimale et nous ajoutons le code suivant :

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

 // Classe application :
 class MyApp : public wxApp
 {
 public:
    // Méthode virtuelle de démarrage de l'application :
     virtual bool OnInit();
 };

 // Notre fenêtre minimale :
 class MyFrame : public wxFrame
 {
 public:
     // Constructeur :
     MyFrame(const wxString& title);

     // 2 handler d'évènements
     void OnQuit(wxCommandEvent& event);
     void OnAbout(wxCommandEvent& event);

 private:
     // la table des évènements
     DECLARE_EVENT_TABLE()
 };

 IMPLEMENT_APP(MyApp)


 // Notre 'main' :
 bool MyApp::OnInit()
 {
     if ( !wxApp::OnInit() )
         return false;

     MyFrame *frame = new MyFrame(_T("Minimal wxWidgets App"));
     frame->Show(true);

     return true;
 }

 // IDs pour nos menus et contrôles :
 enum
 {
     Minimal_Quit = wxID_EXIT,
     Minimal_About = wxID_ABOUT
 };

 // La table des évènements de notre fenêtre :
 BEGIN_EVENT_TABLE(MyFrame, wxFrame)
     EVT_MENU(Minimal_Quit,  MyFrame::OnQuit)
     EVT_MENU(Minimal_About, MyFrame::OnAbout)
 END_EVENT_TABLE()


 // Le constructeur de notre classe de fenêtre :
 MyFrame::MyFrame(const wxString& title)
        : wxFrame(NULL, wxID_ANY, title)
 {
     // Ajoutons nos menus :
     wxMenu *fileMenu = new wxMenu;
     fileMenu->Append(Minimal_Quit, _T("&Quitter\tAlt-Q"), _T("Sortir du programme"));
     wxMenu *helpMenu = new wxMenu;
     helpMenu->Append(Minimal_About, _T("&A propos...\tF1"), _T("Affiche la boîte à propos"));
     // dans une barre de menu :
     wxMenuBar *menuBar = new wxMenuBar();
     menuBar->Append(fileMenu, _T("&Fichier"));
     menuBar->Append(helpMenu, _T("&Aide"));
     SetMenuBar(menuBar);

     // Une barre de statut :
     CreateStatusBar(2);
```

```
    SetStatusText(_T("Bienvenu sur notre premier projet wxWidgets avec MinGW !"));
}


// La gestion des évènements :

void MyFrame::OnQuit(wxCommandEvent& WXUNUSED(event))
{
    Close(true);
}

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    wxMessageBox(wxString::Format(
                    _T("Bienvenu avec %s!\n")
                    _T("\n")
                    _T("Ceci est notre application minimale\n")
                    _T("s'exécutant sous %s."),
                    wxVERSION_STRING,
                    wxGetOsDescription().c_str()
                ),
                _T("A propos..."),
                wxOK | wxICON_INFORMATION,
                this);
}
```

Pour cette application minimale, nous nous sommes contentés de reprendre l'exemple proposé par wxWidgets : "\wxWidgets-2.8.10\samples\minimal\minimal.cpp" que nous avons un peu simplifié pour le rendre plus lisible.

Avant de compiler, il nous reste deux réglages. Commençons par indiquer les bibliothèques wxWidgets nécessaires à la bonne compilation de notre projet. Ici, nous n'avons besoin que de core (libwxmsw28ud_core.a) et base (libwxbase28ud.a). Nous retournons donc dans la boîte de dialogue 'Project Build options', dans l'onglet 'Linker settings', pour la cible debug :

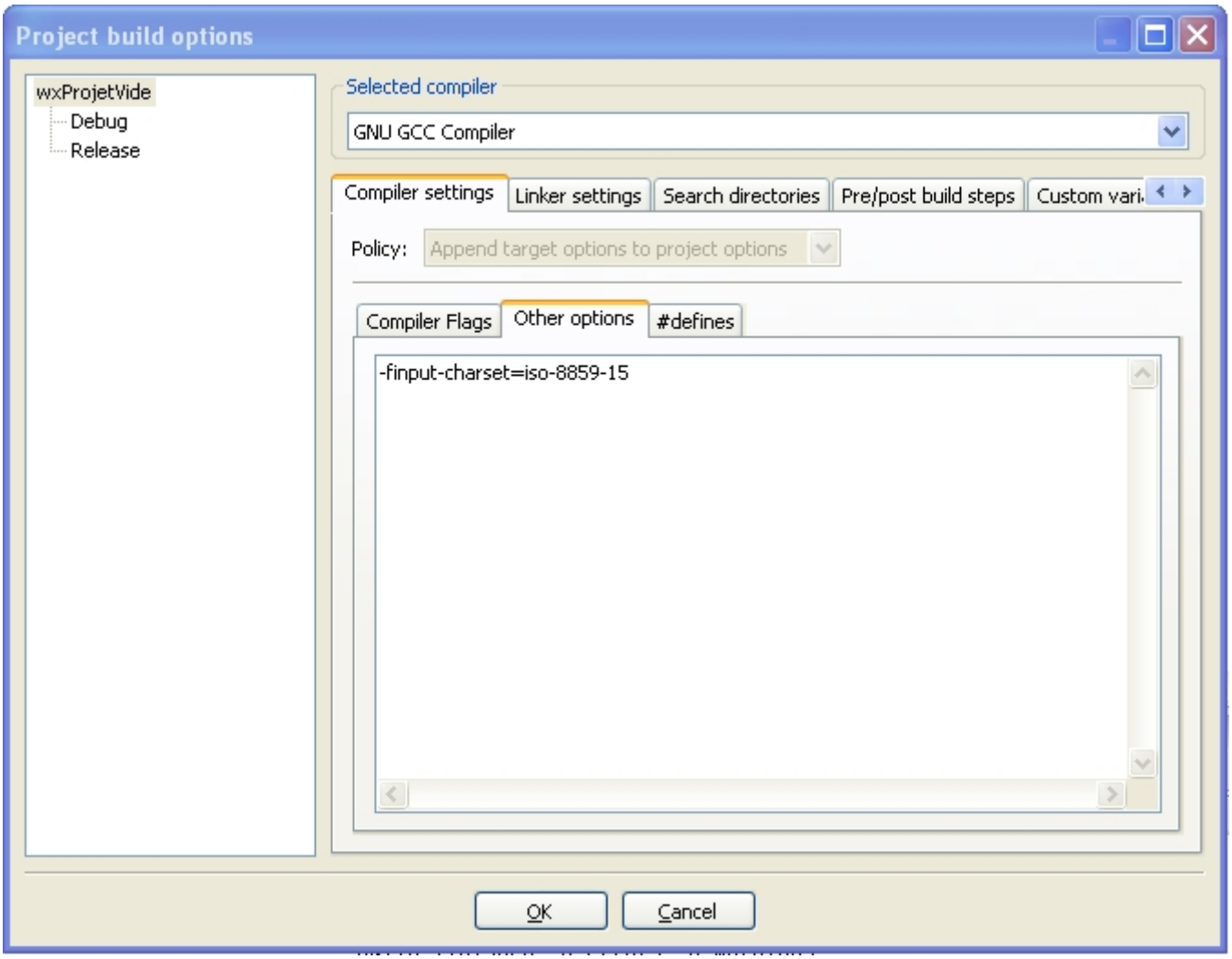


La configuration pour release est identique si ce n'est qu'il faut choisir les versions release de bibliothèques (libwxmsw28u_core.a et libwxbase28u.a).

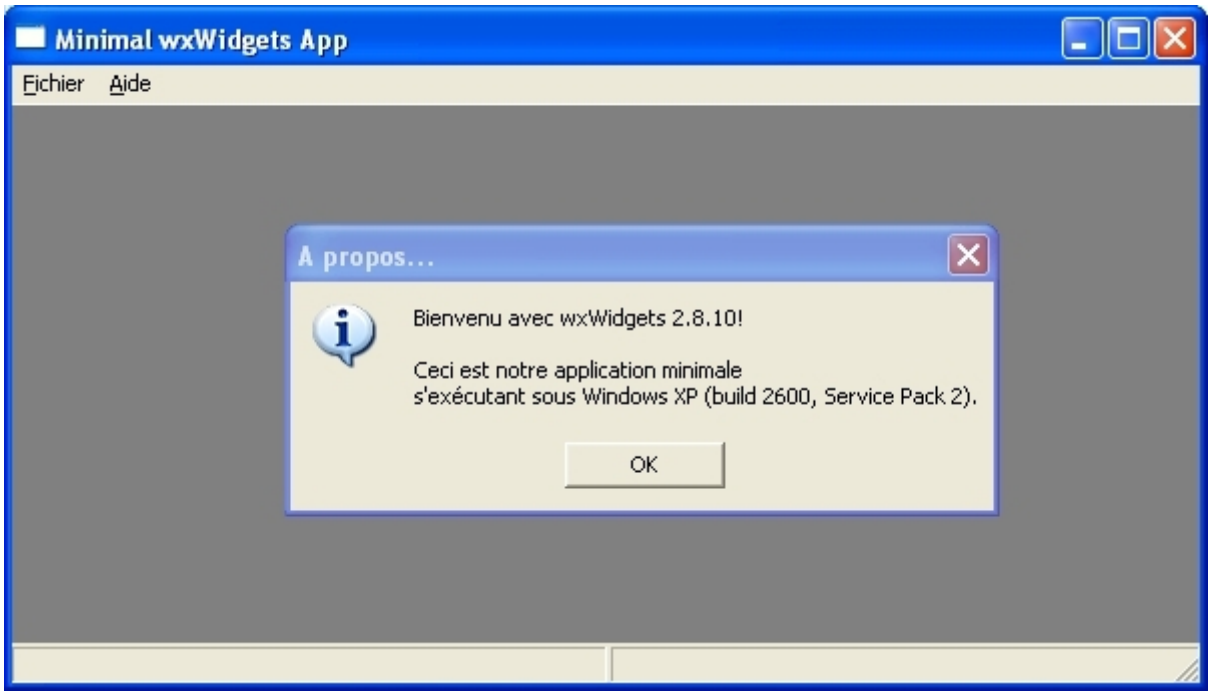
Si nous compilons en l'état notre projet, MinGW génère une erreur :

```
error: converting to execution character set: Illegal byte sequence
```

Cette erreur vient des caractères accentués, tréma, accent circonflexe, c cédille et autres joyeusetés de la langue française. Pour corriger cela nous devons indiquer que notre fichier source est écrit dans notre langue maternelle et que nous souhaitons que la correspondance adéquate soit faite pour la compilation. Il faut rajouter l'option de compilation '-finput-charset=iso-8859-15'. Je vous invite à consulter la documentation de GCC pour des informations plus précises à ce sujet. Cette option s'ajoute dans le sous-onglet 'Other options' de l'onglet 'Compiler settings' pour l'ensemble des cibles de compilation :



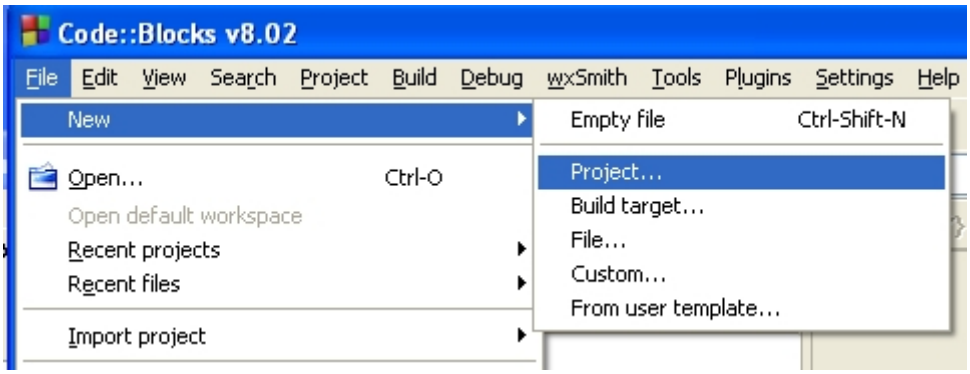
Voilà ! Nous pouvons enfin compiler notre application, demander à l'exécuter et contempler le premier fruit de notre labeur :



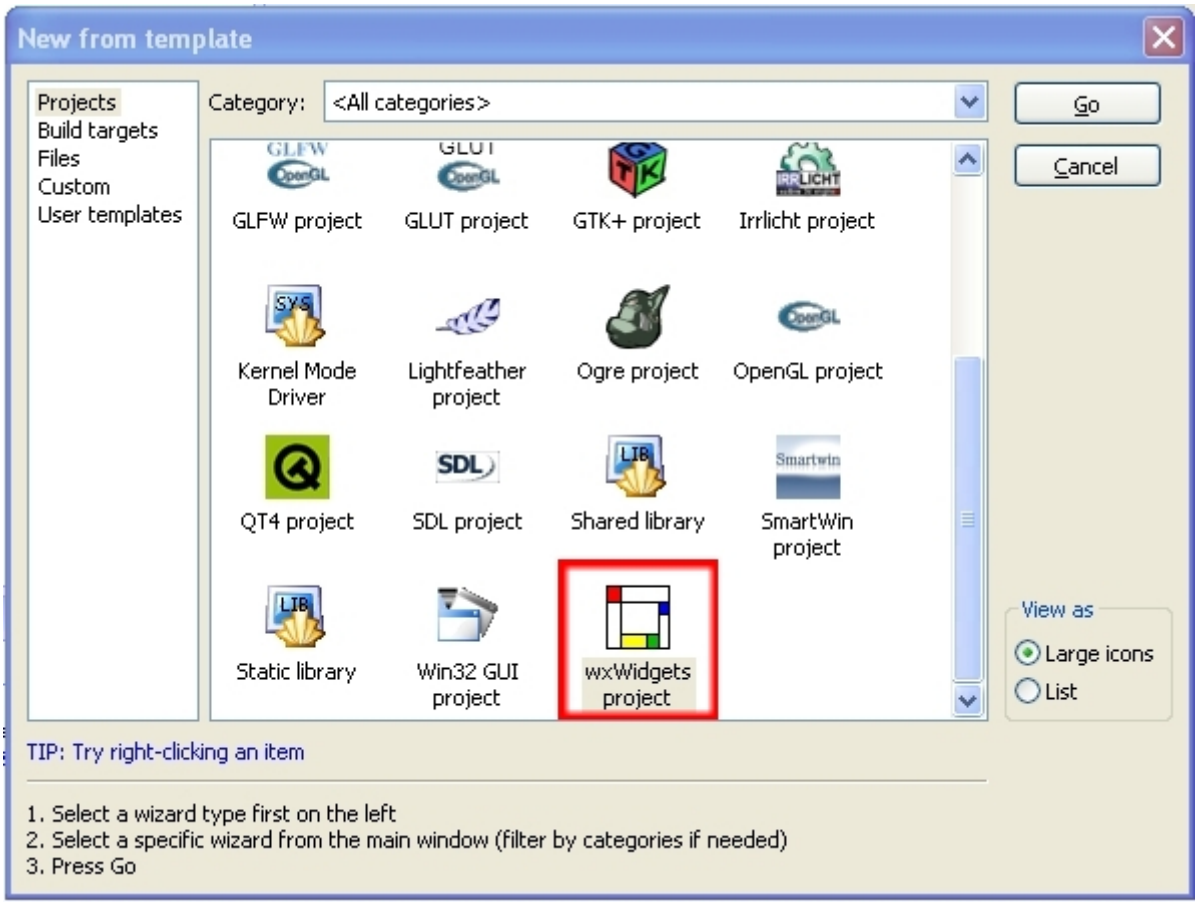
Vous pouvez trouver les sources de ce projet dans l'archive suivante : Source **Projet**

II-C-2 - A partir d'un nouveau projet

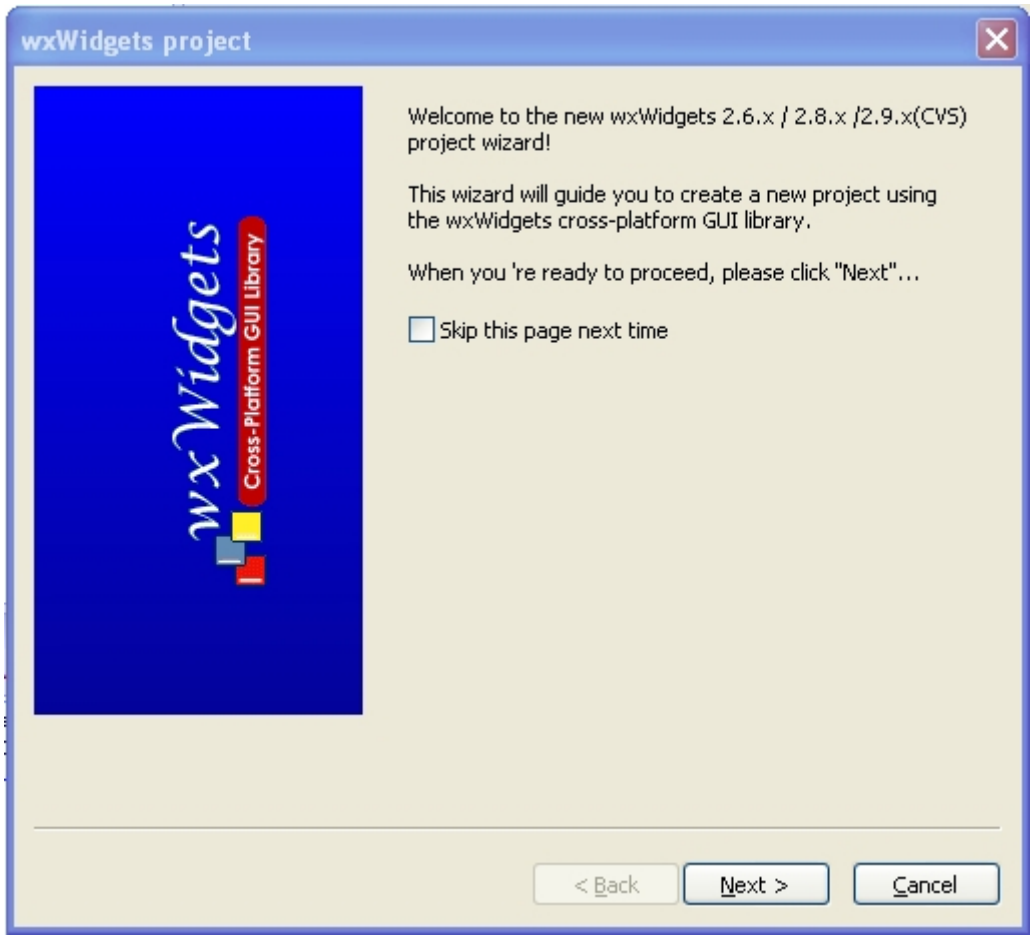
Code::Block propose dans les projets un canevas pour la création de projet wxWidgets prenant ainsi en charge une partie de la configuration. Nous allons montrer comment procéder pour créer un tel projet et le configurer correctement.
Créons notre projet ! Sous Code::Block, nous choisissons le menu 'File/New/Project' :



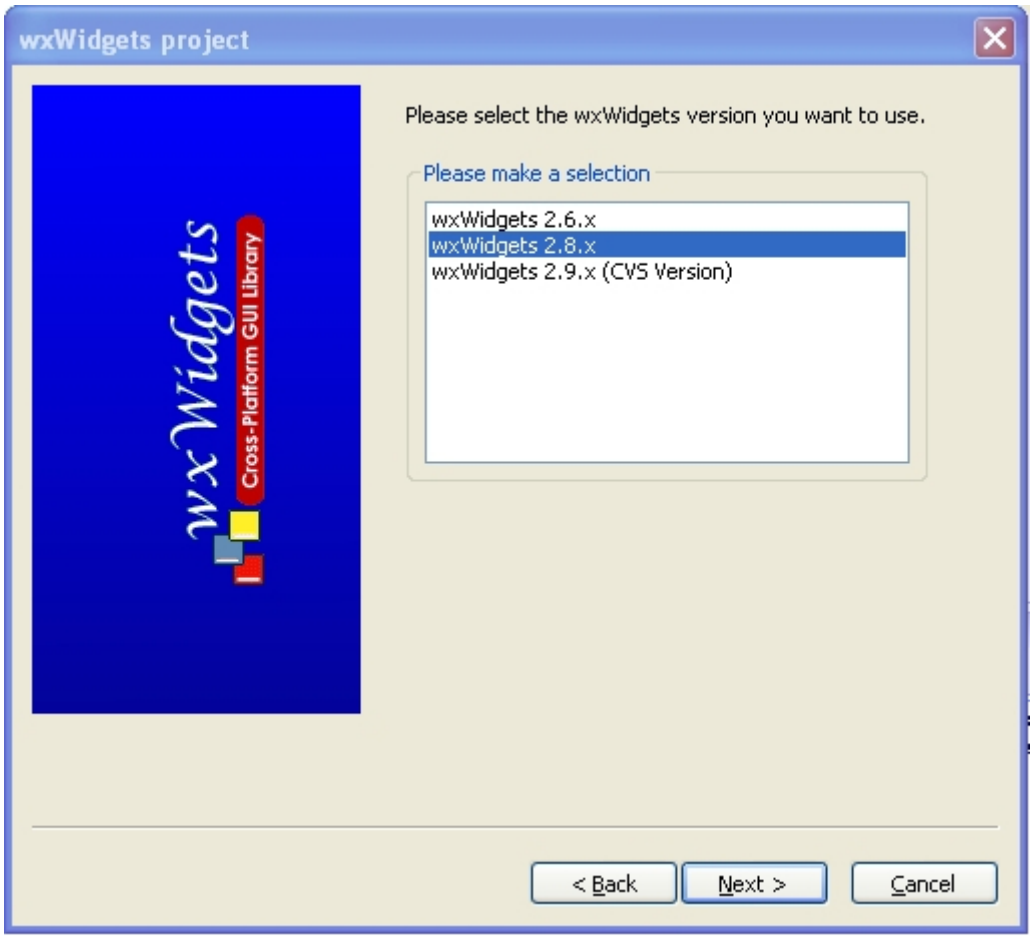
Parmi les différents types de projets, nous optons pour 'wxWidgets project' :



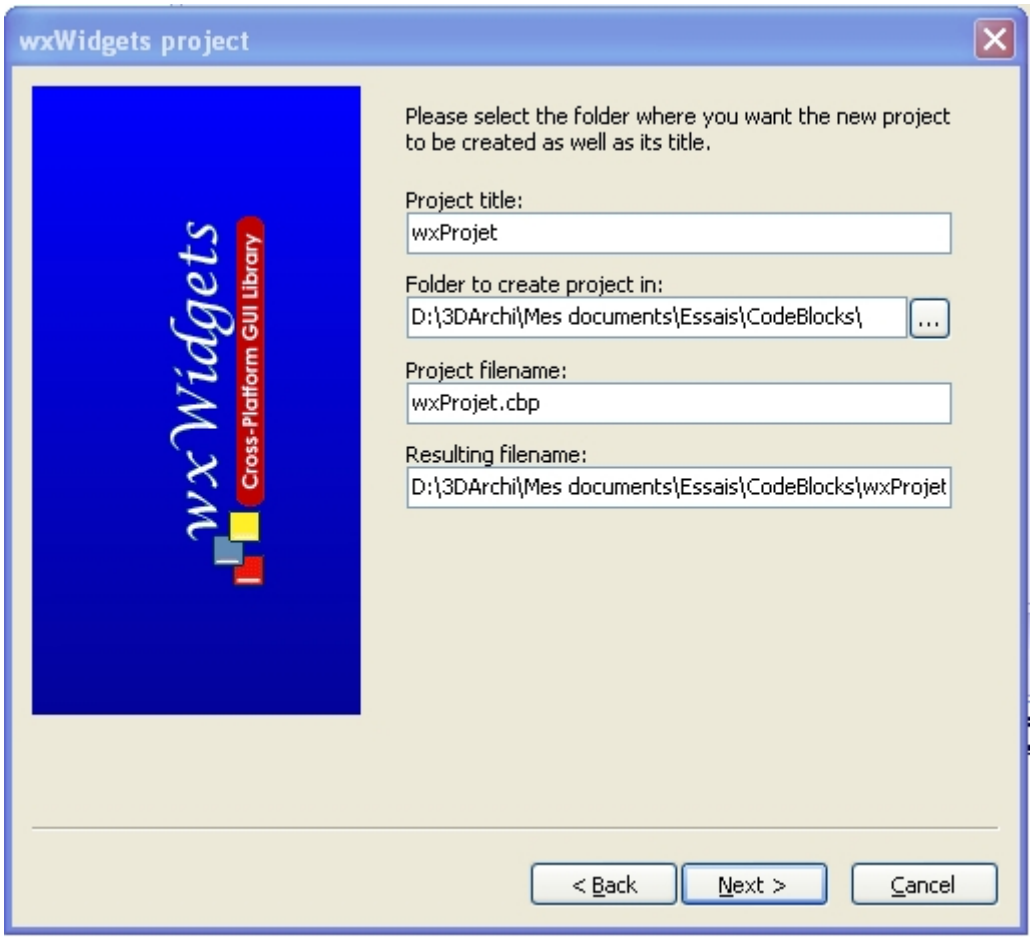
L'assistant démarre :



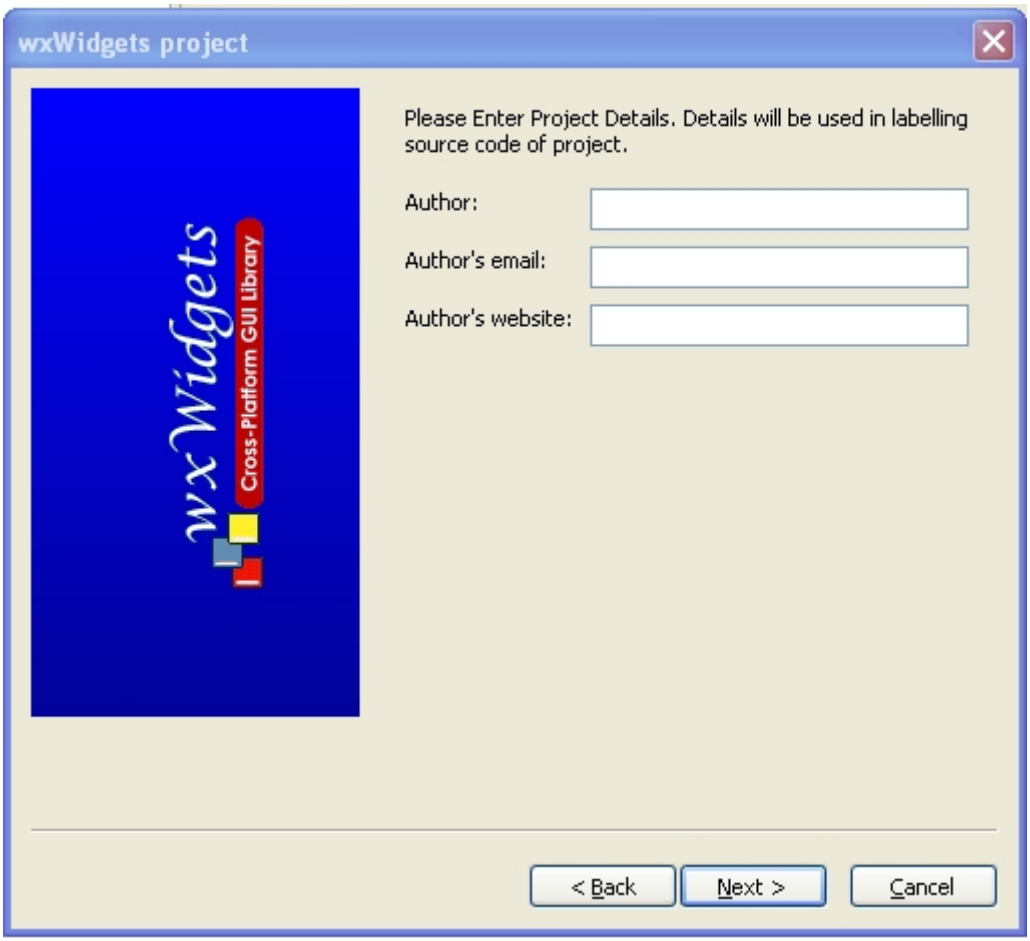
Nous choisissons wxWidgets 2.8.x puisque c'est la version que nous venons d'installer :



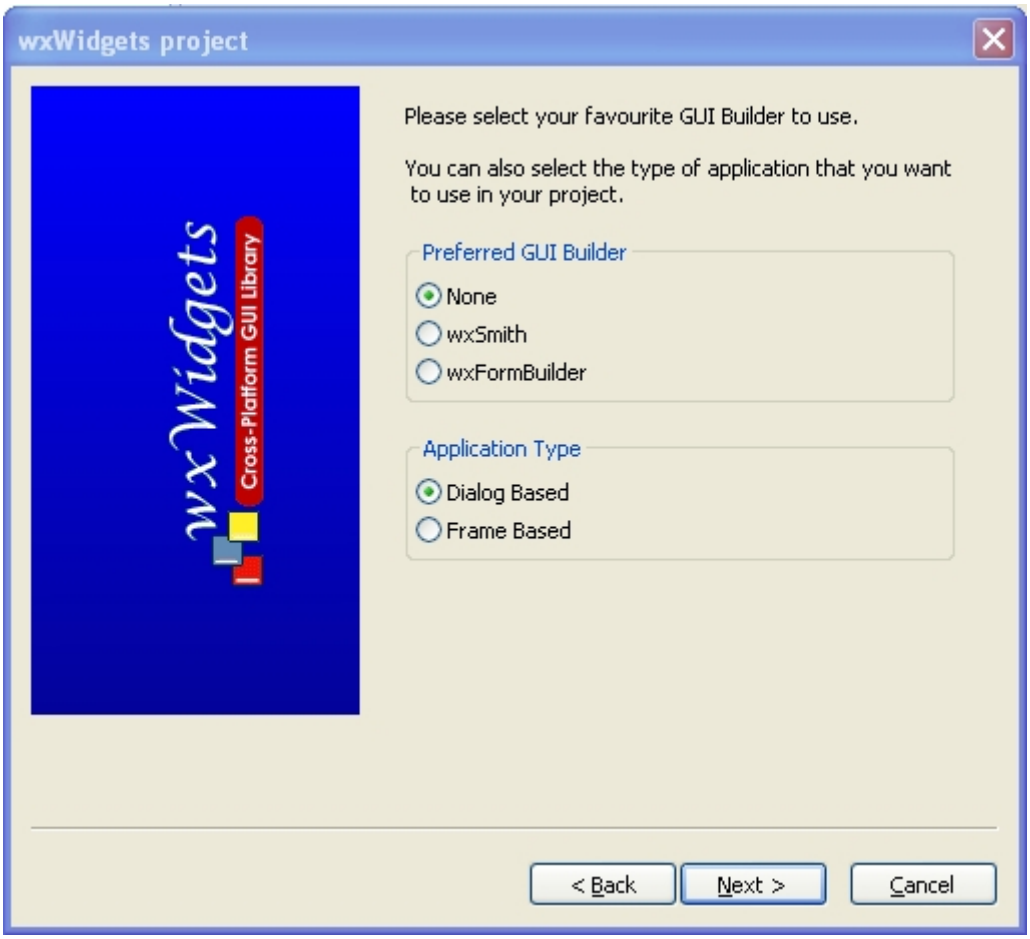
Nous saisissons un nom à notre projet et un répertoire d'accueil :



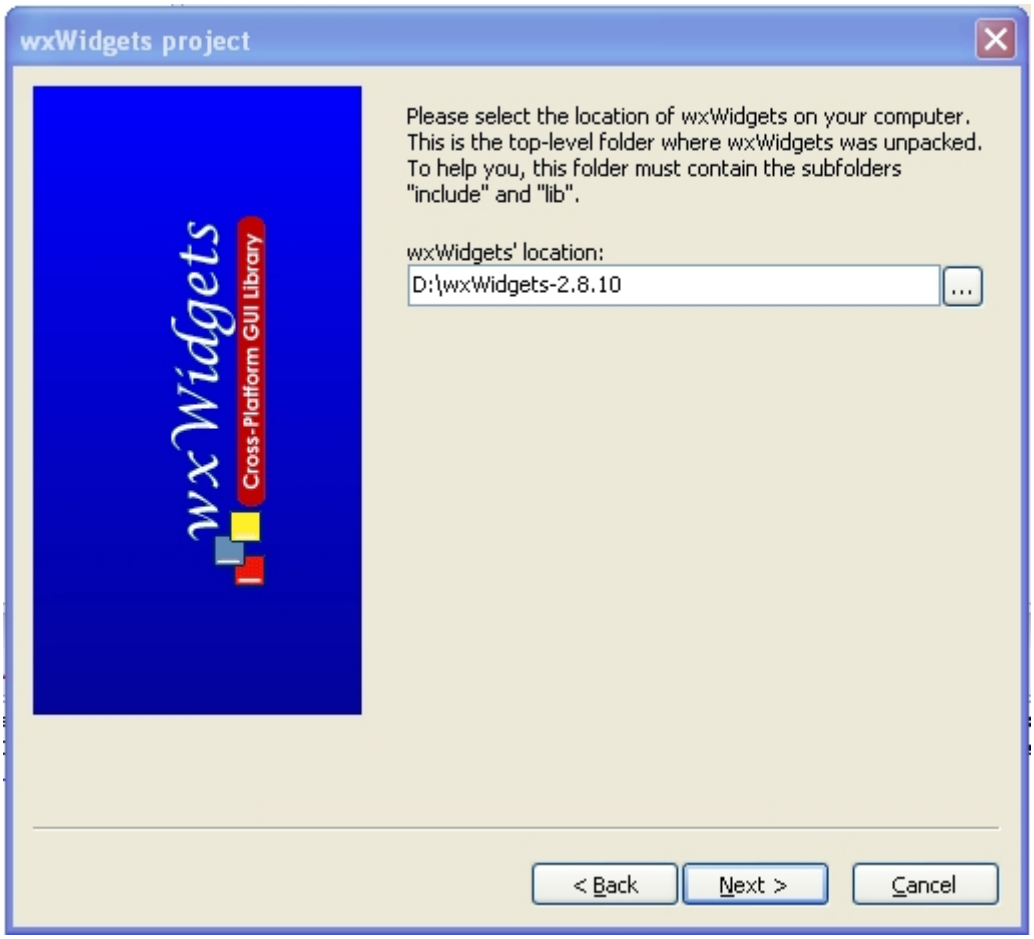
Ces trois informations sont uniquement destinées aux commentaires insérés dans les fichiers sources générés par le canevas :



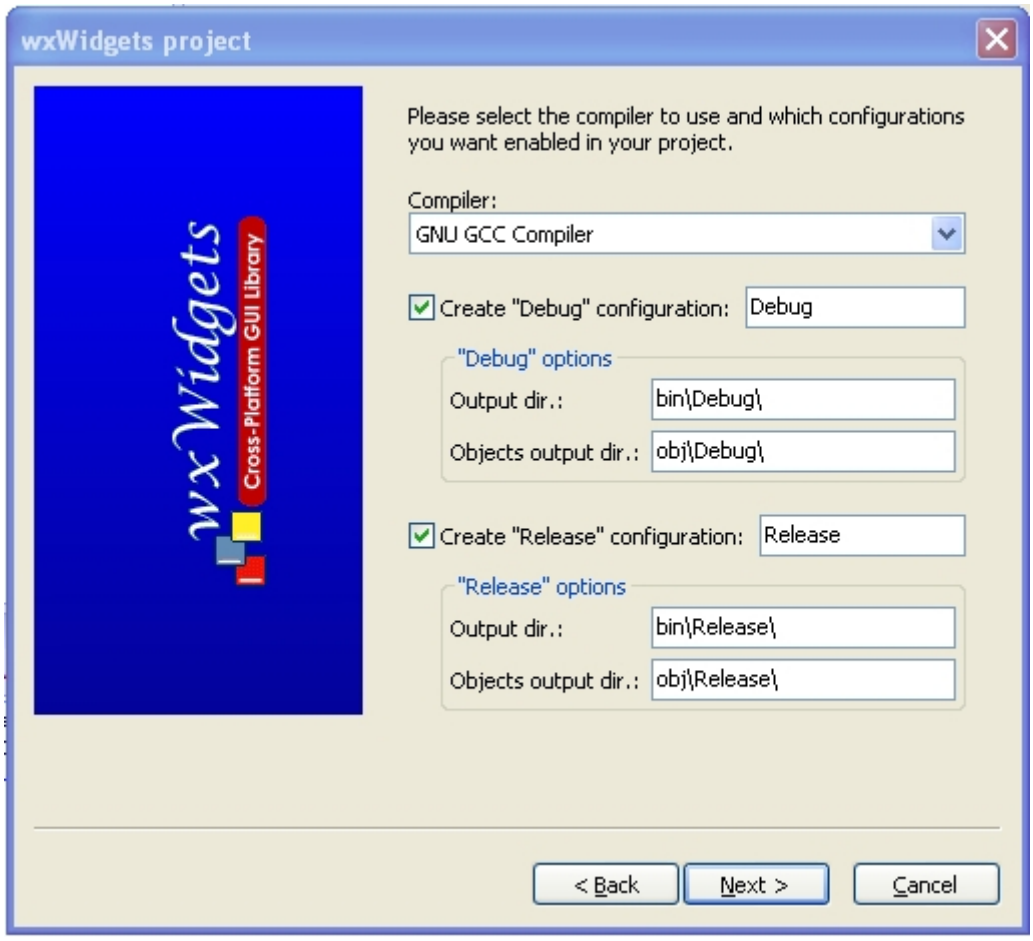
wxSmith et wxFormBuilder sont deux assistants pour la création des I.H.M. wxWidgets. Pour cet exemple, nous n'en choisissons aucun.
Nous créons un projet basé sur une boîte de dialogue :



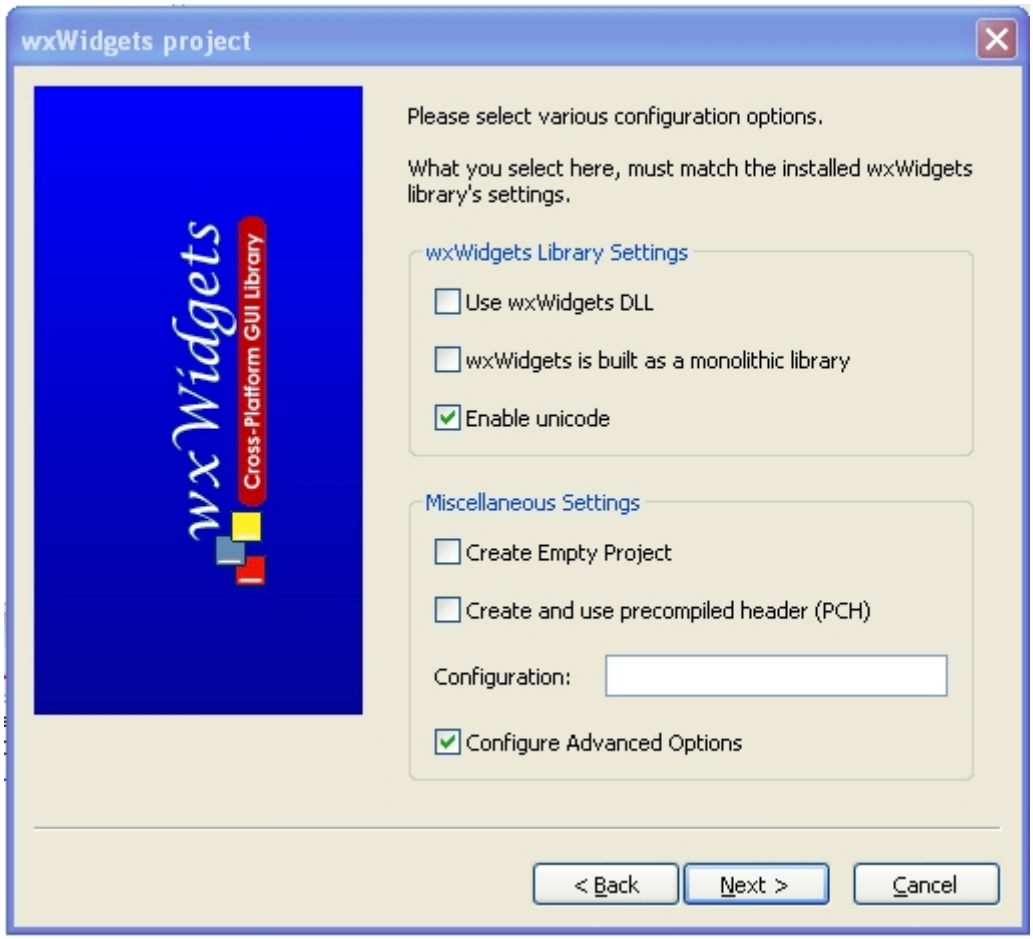
Il faut maintenant indiquer à Code::Block où se trouvent nos fichiers wxWidgets. Code::Block attend le répertoire racine, il en déduit les répertoires d'inclusions des en-têtes et des bibliothèques à partir de cela :



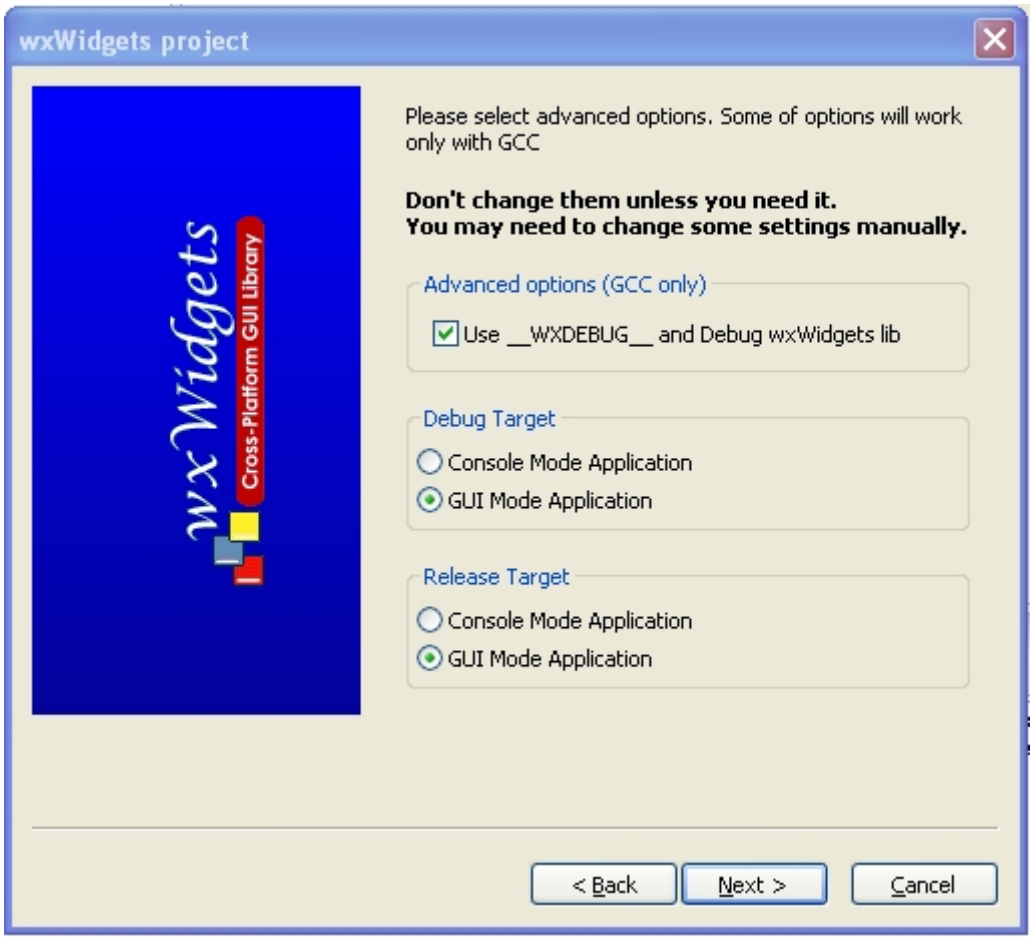
Le panneau suivant nous permet de préciser que nous souhaitons une cible debug et release et que nous allons utiliser le compilateur GCC (nous supposons que vous avez correctement paramétré Code::Block pour aller chercher le compilateur MinGW correspondant) :



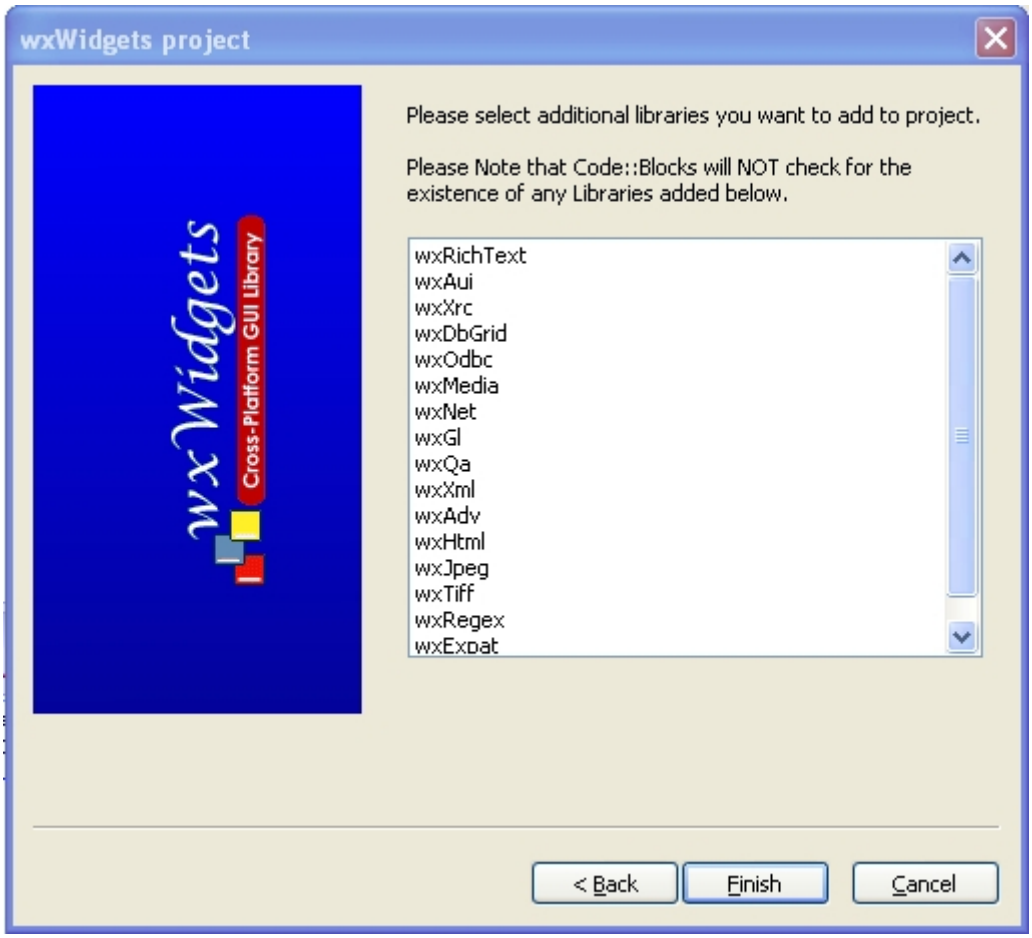
La première case est à cocher lorsque vous avez généré une version DLL de wxWidgets. La seconde si vous avez opté pour une version monolithique et la troisième pour UNICODE. Nos versions sont statiques, en plusieurs unités et en UNICODE : seule la troisième case reste cochée.
Nous ne souhaitons pas créer un projet vide et pour cet exemple nous nous passons des en-têtes précompilés.
N'oubliez pas de cocher la dernière case pour avoir accès aux options avancées :



Nous souhaitons pouvoir utiliser la version debug des bibliothèques en mode debug car nous sommes curieux de ce qu'il se passe : donc nous cochons la case relative à __WXDEBUG__. Notre application est une application IHM en release et en debug :



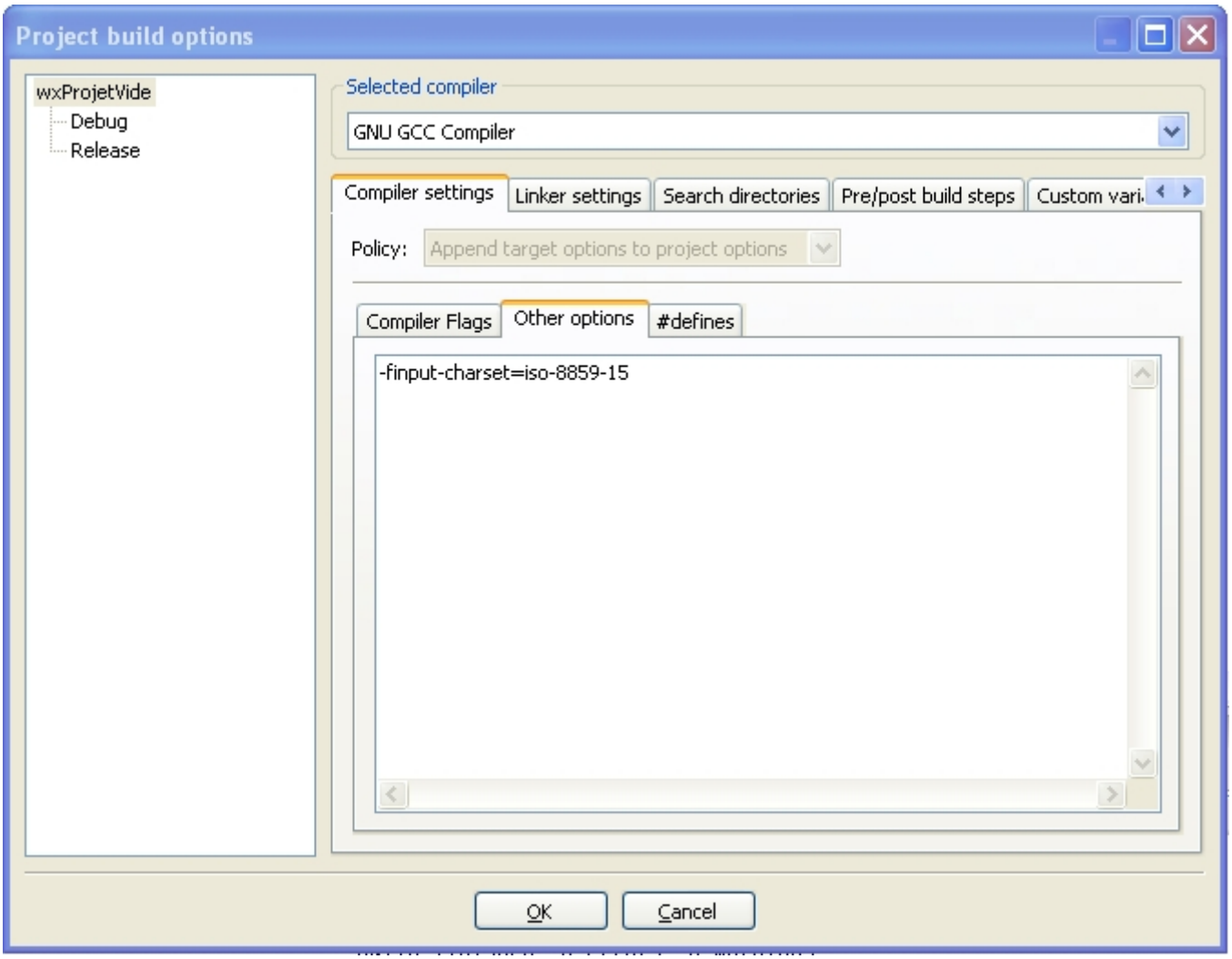
Enfin, cette dernière étape nous permet de préciser quelles sont les bibliothèques avec lesquelles nous souhaitons lier notre application. Par défaut, core (libwxmsw28u[d]_core.a) et base (libwxbase28u[d].a) sont rajoutées. C'est pour ça qu'elles n'apparaissent pas dans la liste. Pour notre exemple simple, nous n'avons pas besoin de bibliothèques supplémentaires :



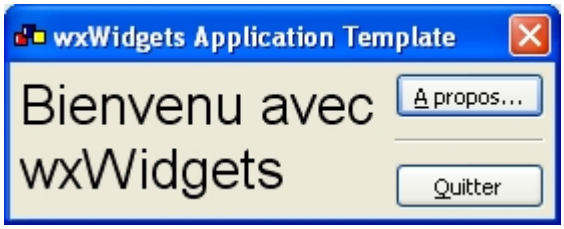
Notre projet est généré. Code::Block a constuit 5 fichiers :

- wxProjetApp.h : fichier de déclaration de notre classe wxProjetApp dérivant de wxApp ;
- wxProjetApp.cpp : fichier de définition de la classe wxProjetApp ;
- wxProjetMain.h : fichier de déclaration de notre classe wxProjetDialog dérivant de wxDialog ;
- wxProjetMain.cpp : fichier de définition de la classe wxProjetDialog ;
- resource.rc : ficher de ressources (icônes).

Avant de lancer la compilation, nous devons positionner l'option '-finput-charset=iso-8859-15'. Je vous invite à consulter la documentation de GCC pour des informations plus précises à ce sujet. Cette option s'ajoute dans le sous-onglet 'Other options' de l'onglet 'Compiler settings' pour l'ensemble des cibles de compilation :



Nous pouvons maintenant lancer la compilation et exécuter notre programme :



Vous pouvez trouver les sources de ce projet dans l'archive suivante : Source **Projet**

II-D - Compiler les fichiers exemples

Le répertoire "\wxWidgets-2.8.10\samples\" regroupe des projets d'exemples illustrant différentes classes de wxWidgets. Chaque répertoire suit le même principe. Il contient le ou les fichiers sources ainsi que le makefile nécessaire à la compilation pour chaque compilateur. Générer des exemples devient chose facile.
Placez-vous dans le répertoire \wxWidgets-2.8.10\samples\access et tapez simplement la commande suivante :

```
mingw32-make -f makefile.gcc BUILD=debug
```

Cette commande ressemble en tout point à celle utilisée pour construire les bibliothèques.
La compilation d'un fichier exemple entraîne la création d'un répertoire gcc_mswud contenant l'exécutable généré.

Il est possible de générer tous les exemples en une seule fois. Il suffit de se positionner sur le répertoire "\wxWidgets-2.8.10\samples". Celui-ci contient un makefile.gcc pilotant la construction de tous les exemples. Il ne nous reste donc qu'à demander cette compilation :

```
mingw32-make -f makefile.gcc BUILD=debug
```

II-E - Pour les lecteurs pressés : résumé

1 **Télécharger wxWidgets**
2 Désarchiver wxWidgets (répertoire destination sans espace)
3 Modifier la configuration : \wxWidgets-2.8.10\build\msw\config.gcc :

```
UNICODE ?= 1
USE_OPENGL ?= 1
USE_ODBC ?= 1
```

4 Modifier le setup : \wxWidgets-2.8.10\include\wx\msw\setup.h :

```
#define wxUSE_UNICODE 1
#define wxUSE_GLCANVAS       1
#define wxUSE_ODBC          1
#define wxUSE_STL 1
#define wxUSE_STD_IOSTREAM  1
```

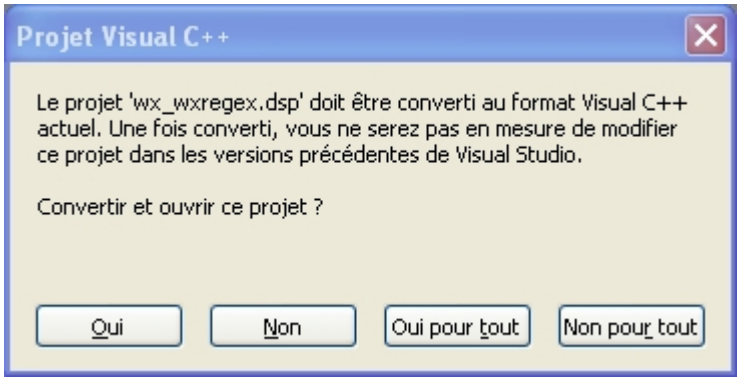
5 Compiler

```
cd \wxWidgets-2.8.10\build\msw\
mingw32-make -f makefile.gcc BUILD=debug
```

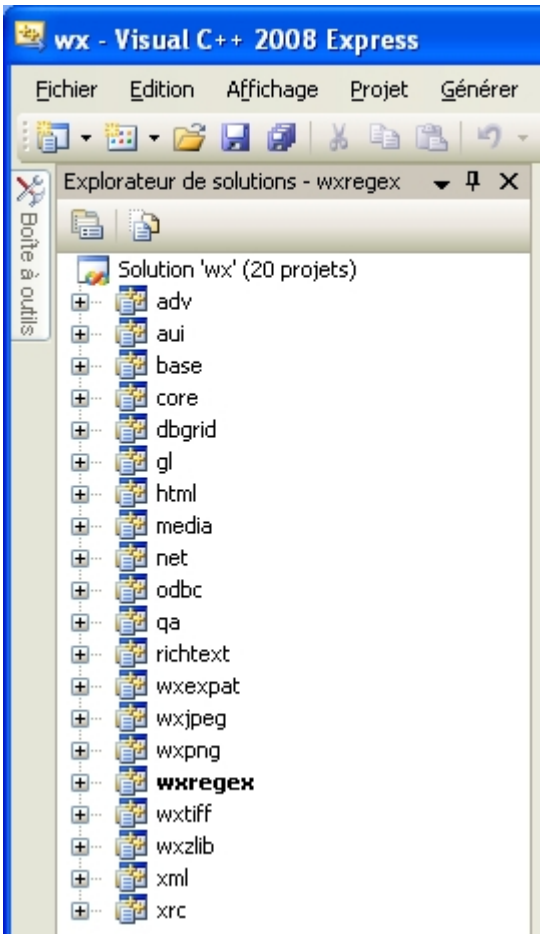
6 Paramétrer Code::Block
- 'Project'/'Build Options...' - Item 'Projet' (racine arbre de gauche) - Onglet 'Search directories' - Sous-onglet 'Compiler' - Ajout de "\wxWidgets-2.8.10\include"
- 'Project'/'Build Options...' - Item 'Projet' (racine arbre de gauche) - Onglet 'Search directories' - Sous-onglet 'Linker' - Ajout de "\wxWidgets-2.8.10\lib\gcc_lib"
- 'Project'/'Build Options...' - Item 'Debug' (arbre de gauche) - Onglet 'Search directories' - Sous-onglet 'Compiler' - Ajout de "\wxWidgets-2.8.10\lib\gcc_lib\mswud" et modification de Policy vers 'Prepend target options to project options'
- 'Project'/'Build Options...' - Item 'Debug' (arbre de gauche) - Onglet 'Search directories' - Sous-onglet 'Compiler' - Ajout de "\wxWidgets-2.8.10\lib\gcc_lib\mswu" et modification de Policy vers 'Prepend target options to project options'

III - wxWidgets et Visual C++ Express

wxWidgets propose deux moyens pour compiler avec Visual C++ : soit en ligne de commande soit avec des projets Visual. Nous allons nous intéresser à la compilation avec les projets Visual déjà définis.
Ouvrer le fichier "\wxWidgets-2.8.10\build\msw\wx.dsw". Les fichiers projets datent de la version Visual Studio 6. La version Visual Express 2008 propose de convertir les fichiers au nouveau format Visual C++ :



Répondez 'Oui pour tout'.
Visual convertit et charge tous les projets associés. Vous obtenez la solution complète :



Les fichiers relatifs aux nouveaux projets sont créés dans le répertoire "\wxWidgets-2.8.10\build\msw\" : l'ensemble des fichiers .vcproj pour chacun des projets ainsi que le .sln pour la solution.

III-A - Paramétrer la compilation

Différentes configurations sont déjà définies pour les compilations. Elles suivent le schéma suivant :

```
[DLL] [Universal] [Unicode] [Debug|Release]
```

La présence de 'DLL' indique la version SHARED, son absence la version STATIC.
'Universal' choisit le rendu wxWidgets. Sans cela, le rendu MSW est produit.
'Unicode' pilote la version UNICODE.
'Debug' permet la compilation de la version DEBUG et 'Release' la version RELEASE.
Nous avons choisi de générer les versions DEBUG et RELEASE des bibliothèques STATIC avec UNICODE, le port MSW et en plusieurs bibliothèques. Nous sommes donc intéressés par les configurations :

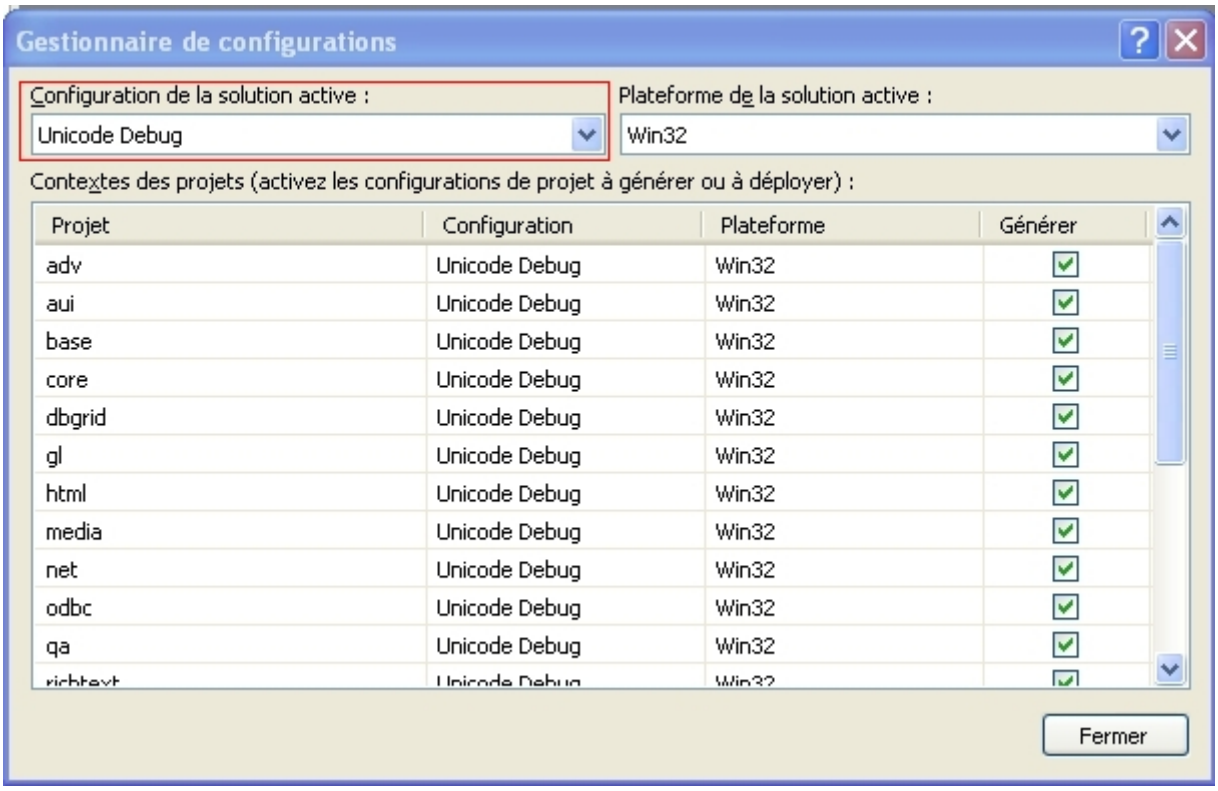
```
Unicode Debug
```

```
Unicode Release
```

Le choix de la configuration s'opère directement dans la liste déroulante de la barre d'outils en haut de l'environnement de développement :

La configuration peut aussi être positionnée en choisissant l'item 'Gestionnaire de configurations...' du menu 'Générer' :

Une boîte de dialogue apparaît avec la liste de tous les projets. La configuration est choisie dans la liste déroulante 'Configuration de la solution active :'



La modification de la configuration active dans cette liste déroulante modifie automatiquement la configuration de chaque projet. Ne modifier en aucun cas individuellement la configuration d'un des projets. La case à cocher indique si le projet doit être généré (case cochée) ou non (case non cochée). Laissez toutes les cases cochées.

Une fois notre configuration choisie, nous devons également intervenir sur le fichier "\wxWidgets-2.8.10\include\wx\msw\setup.h".
En l'occurrence, nous allons modifier les directives suivantes :

- #define wxUSE_UNICODE 1 : car nous voulons compiler avec UNICODE
- #define wxUSE_GLCANVAS 1 : car nous voulons compiler avec OpenGL
- #define wxUSE_ODBC 1 : car nous voulons compiler avec OBDC

A cela, nous modifions les directives suivantes :

- #define wxUSE_STL 1 : wxWidgets définit ses propres classes pour les listes (wxList) et les tableaux (wxArray). En positionnant cette directive à 1, ces deux classes vont dériver de std::list et std::vector tirant ainsi profit de la STL.
- #define wxUSE_STD_IOSTREAM 1 : grâce à cette directive, nous indiquons que nous souhaitons utiliser les flux de la bibliothèque standard quand cela est possible à la place des flux définis dans wxWidgets.

III-B - Compiler

Après avoir choisi la configuration 'Unicode Debug', la compilation se lance tout simplement avec l'item de menu 'Générer la solution' du menu 'Générer' :

Les fichiers intermédiaires de la compilation (les .obj) sont générés dans "\wxWidgets-2.8.10\build\msw\vc_mswud \XXX", où XXX est le projet correspondant. Le nom du répertoire intermédiaire est composé à partir de :

- le compilateur : 'vc' ;
- la plateforme : 'msw' ;
- le mode DEBUG/RELEASE : 'd' pour debug, rien pour release ;
- le mode UNICODE : 'u' pour UNICODE.

Si nous avions choisi le mode non UNICODE, le répertoire aurait eu la forme "vc_mswd". Et comme nous allons le voir, la compilation en mode RELEASE crée le répertoire "vc_mswu".

La compilation a aussi ajouté un autre répertoire : "\wxWidgets-2.8.10\lib\vc_lib". Le schéma suivi pour le nom du répertoire ne prend en compte que le nom du compilateur utilisé. Toutes les versions des bibliothèques compilées avec un même compilateur (DEBUG, RELEASE, UNICODE, ANSI) sont mises dans ce même répertoire ce qui facilite le paramétrage de Visual. Ce répertoire contient à sa base l'ensemble des bibliothèques générées : les fichiers .lib et les fichiers pdb (information de debug) et idb (pour la recompilation minimale). Les bibliothèques générées sont :

- wxbase28ud.lib
- wxbase28ud_net.lib
- wxbase28ud_odbc.lib
- wxbase28ud_xml.lib
- wxexpatd.lib
- wxjpegd.lib
- wxmsw28ud_adv.lib
- wxmsw28ud_aui.lib
- wxmsw28ud_core.lib
- wxmsw28ud_dbgrid.lib
- wxmsw28ud_gl.lib
- wxmsw28ud_html.lib
- wxmsw28ud_media.lib
- wxmsw28ud_qa.lib
- wxmsw28ud_richtext.lib
- wxmsw28ud_xrc.lib
- wxpngd.lib
- wxregexud.lib
- wxtiffd.lib
- wxzlibd.lib

Chaque bibliothèque porte un nom suivant le schéma :

```
[nom]ud.lib
```

Certaines n'ont pas le suffixe 'u' dès lors qu'elles ne dépendent pas du critère UNICODE.
Toujours dans "\wxWidgets-2.8.10\lib\vc_lib" un répertoire a été créé avec les bibliothèques : "\wxWidgets-2.8.10\lib\vc_lib\mswud\wx" avec en son sein un fichier "setup.h". Il s'agit du fichier d'en-tête correspondant à la compilation. Nous verrons plus loin qu'il est important de bien configurer Visual de façon à ce que chaque ligne *#include <wx/setup.h>* corresponde à l'inclusion de ce fichier d'en-tête et non pas à celui dans "\wxWidgets-2.8.10\include\wx\msw\setup.h".

De façon identique, la compilation en RELEASE s'obtient en choisissant la configuration 'Unicode Release' et en demandant la génération de la solution.

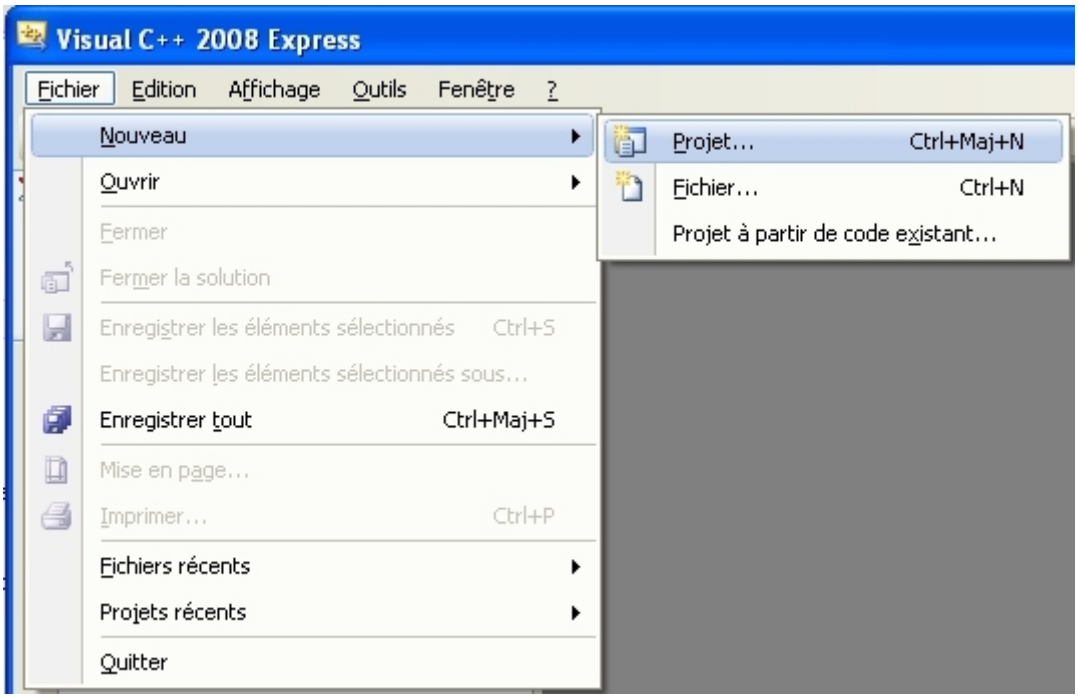
Les fichiers intermédiaires sont produits dans le répertoire "wxWidgets-2.8.10\build\msw\vc_mswu".
Les bibliothèques générées dans "\wxWidgets-2.8.10\lib\vc_lib" sont les mêmes que précédemment mais sans le suffixe 'd' pour DEBUG :

- wxbase28u.lib
- wxbase28u_net.lib
- wxbase28u_odbc.lib
- wxbase28u_xml.lib
- wxexpat.lib
- wxjpeg.lib
- wxmsw28u_adv.lib
- wxmsw28u_aui.lib
- wxmsw28u_core.lib
- wxmsw28u_dbgrid.lib
- wxmsw28u_gl.lib
- wxmsw28u_html.lib
- wxmsw28u_media.lib
- wxmsw28u_qa.lib
- wxmsw28u_richtext.lib
- wxmsw28u_xrc.lib
- wxpng.lib
- wxregexu.lib
- wxtiff.lib
- wxzlib.lib

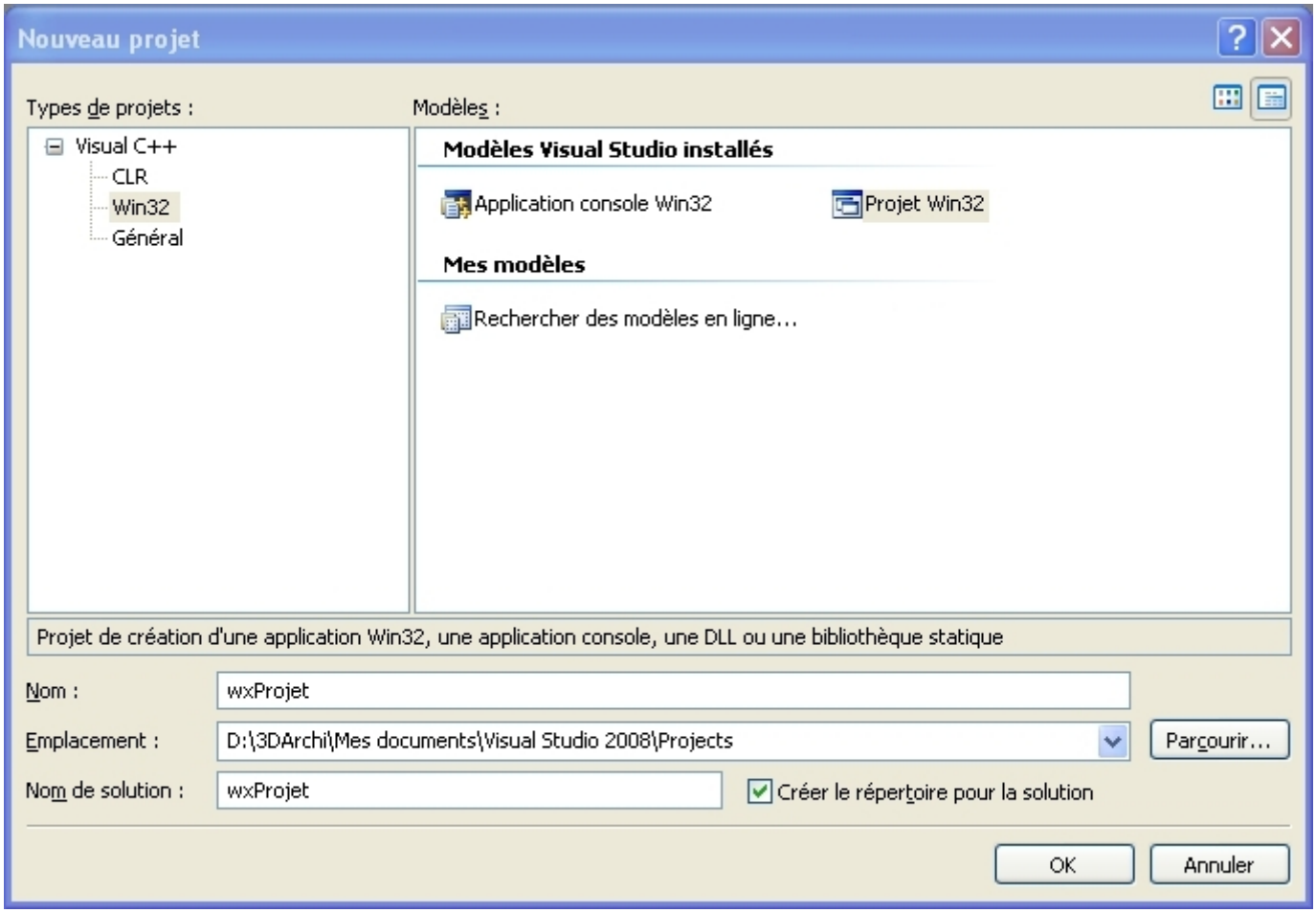
Le répertoire "\wxWidgets-2.8.10\lib\vc_lib\mswu" contient le répertoire "wx" avec son fichier "setup.h".

III-C - Paramétrer Visual C++

Commençons par créer un projet. Pour cela, nous choisissons l'item de menu 'Projet' du sous-menu 'Nouveau' du menu 'Fichier' :

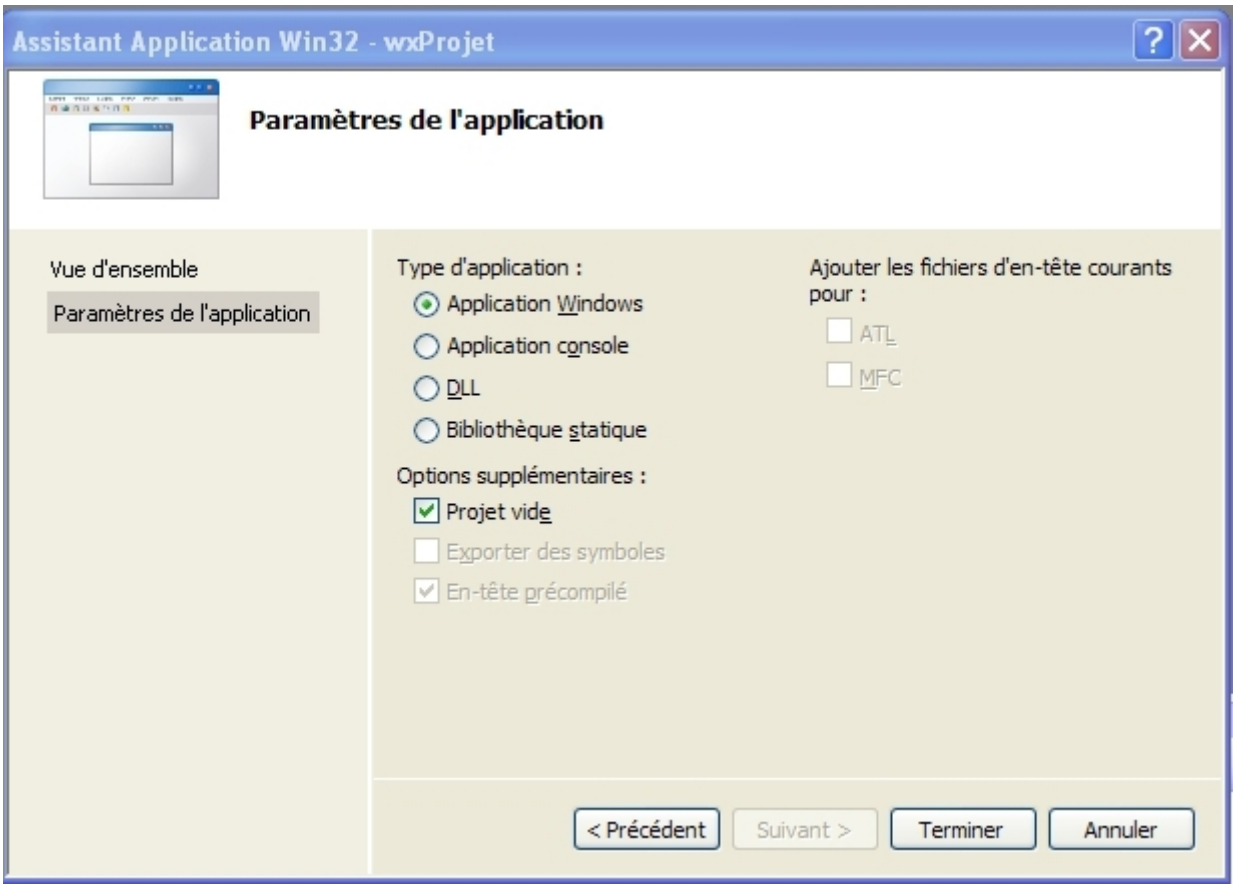


La boîte de dialogue suivante nous permet de saisir le type de projet (Projet Win32 parmi les modèles Win32), le nom du projet, son emplacement, la solution générée :

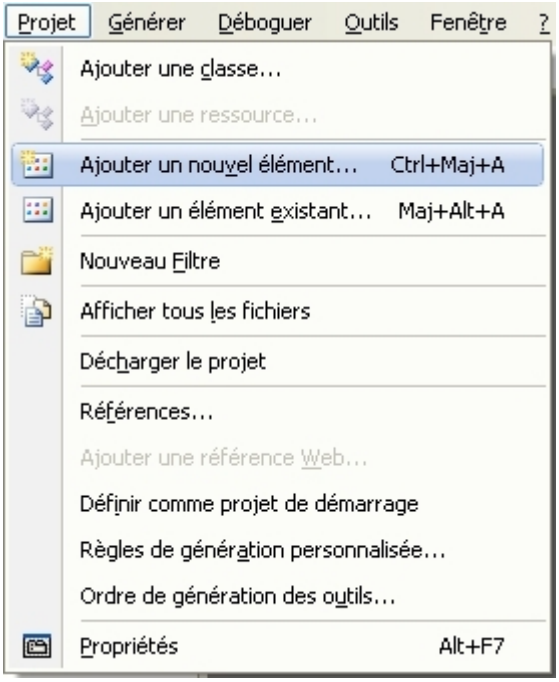


L'assistant de création de projet apparaît :

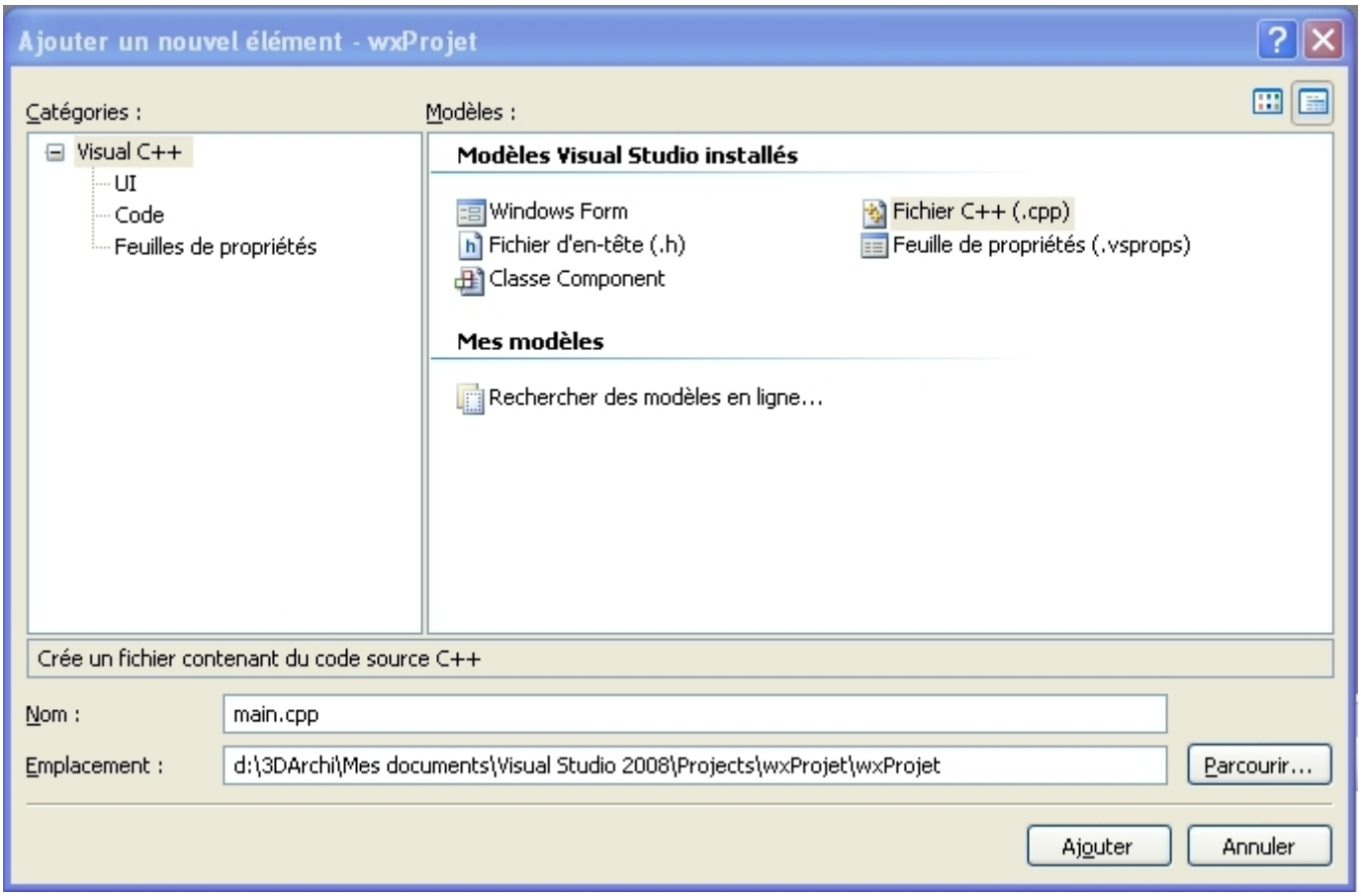
Après avoir cliqué sur suivant, nous positionnons les paramètres de notre projet 'Application Windows' et choisissons un projet vide :



Pour notre fichier source, nous choisissons l'item de menu 'Ajouter un nouvel élément...' du menu 'Projet' :



Dans la boîte de dialogue, optons pour un 'Fichier C++' et donnons-lui son nom 'main.cpp' :



Copions le code suivant et sauvegardons le fichier sous main.cpp :

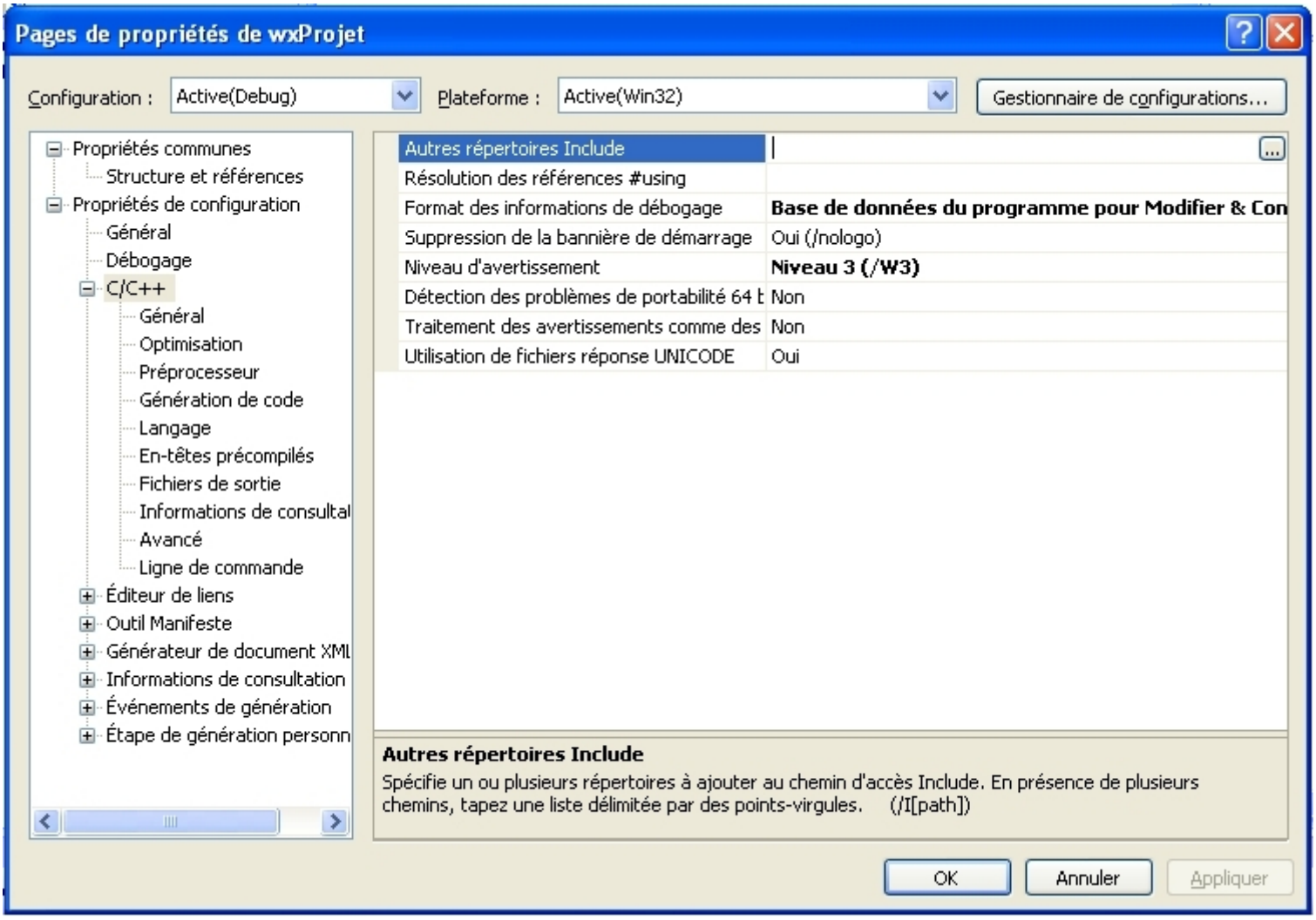
```
#include "wx/wx.h"

 // Classe application :
 class MyApp : public wxApp
 {
 public:
    // Méthode virtuelle de démarrage de l'application :
     virtual bool OnInit();
 };

 // Notre fenêtre minimale :
 class MyFrame : public wxFrame
 {
 public:
     // Constructeur :
     MyFrame(const wxString& title);

     // 2 handler d'évènements
     void OnQuit(wxCommandEvent& event);
     void OnAbout(wxCommandEvent& event);

 private:
     // la table des évènements
     DECLARE_EVENT_TABLE()
 };

 IMPLEMENT_APP(MyApp)


 // Notre 'main' :
 bool MyApp::OnInit()
 {
     if ( !wxApp::OnInit() )
         return false;
```

```
    MyFrame *frame = new MyFrame(_T("Minimal wxWidgets App"));
    frame->Show(true);

    return true;
}

// IDs pour nos menus et contrôles :
enum
{
    Minimal_Quit = wxID_EXIT,
    Minimal_About = wxID_ABOUT
};

// La table des évènements de notre fenêtre :
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(Minimal_Quit,  MyFrame::OnQuit)
    EVT_MENU(Minimal_About, MyFrame::OnAbout)
END_EVENT_TABLE()


// Le constructeur de notre classe de fenêtre :
MyFrame::MyFrame(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title)
{
    // Ajoutons nos menus :
    wxMenu *fileMenu = new wxMenu;
    fileMenu->Append(Minimal_Quit, _T("&Quitter\tAlt-Q"), _T("Sortir du programme"));
    wxMenu *helpMenu = new wxMenu;
    helpMenu->Append(Minimal_About, _T("&A propos...\tF1"), _T("Affiche la boîte à propos"));
    // dans une barre de menu :
    wxMenuBar *menuBar = new wxMenuBar();
    menuBar->Append(fileMenu, _T("&Fichier"));
    menuBar->Append(helpMenu, _T("&Aide"));
    SetMenuBar(menuBar);

    // Une barre de statut :
    CreateStatusBar(2);
    SetStatusText(_T("Bienvenu sur notre premier projet wxWidgets avec MinGW !"));
}


// La gestion des évènements :

void MyFrame::OnQuit(wxCommandEvent& WXUNUSED(event))
{
    Close(true);
}

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    wxMessageBox(wxString::Format(
                    _T("Bienvenu avec %s!\n")
                    _T("\n")
                    _T("Ceci est notre application minimale\n")
                    _T("s'exécutant sous %s."),
                    wxVERSION_STRING,
                    wxGetOsDescription().c_str()
                ),
                _T("A propos..."),
                wxOK | wxICON_INFORMATION,
                this);
}
```

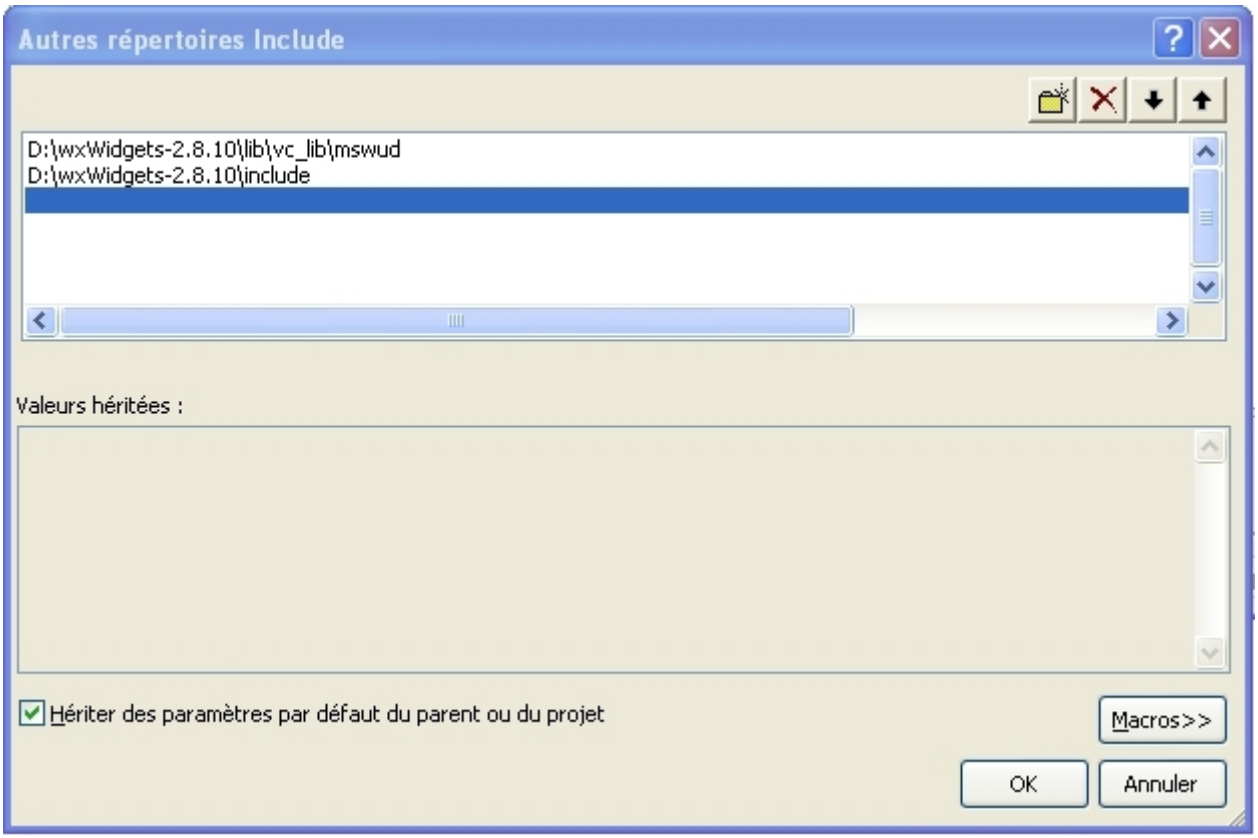
Pour cette application minimale, nous nous sommes contentés de reprendre l'exemple proposé par wxWidgets : "\wxWidgets-2.8.10\samples\minimal\minimal.cpp" que nous avons un peu simplifié pour le rendre plus lisible.

Il faut maintenant configurer correctement notre nouveau projet. La configuration d'un projet avec Visual C++ se modifie en allant chercher l'item 'Propriétés de wxProjet' du menu Projet :

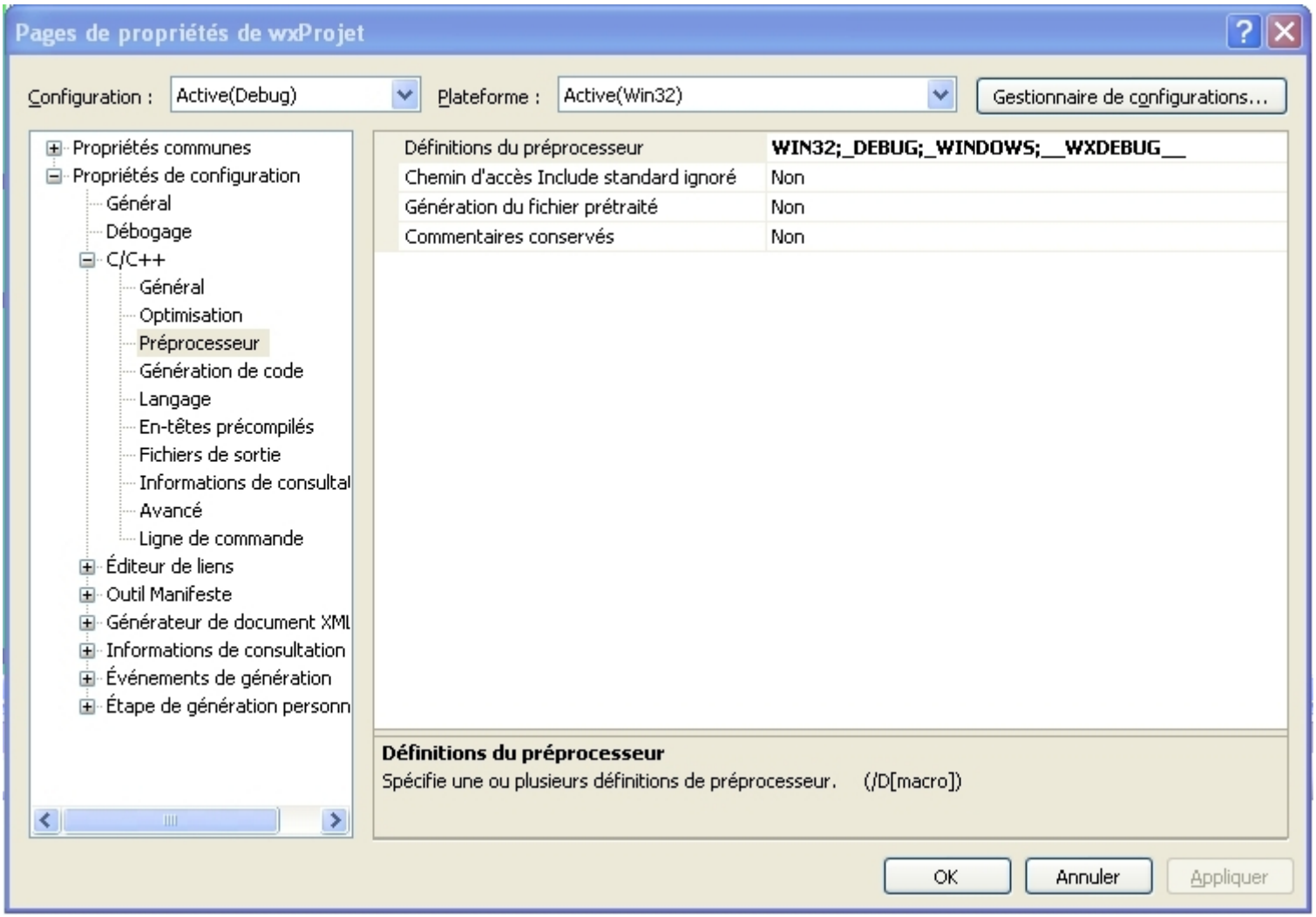
Dans l'arbre de gauche, nous nous positionnons sur 'Propriétés de configuration', 'C/C++' et éditons la première ligne dans le tableau de droite 'Autres répertoires Include' :



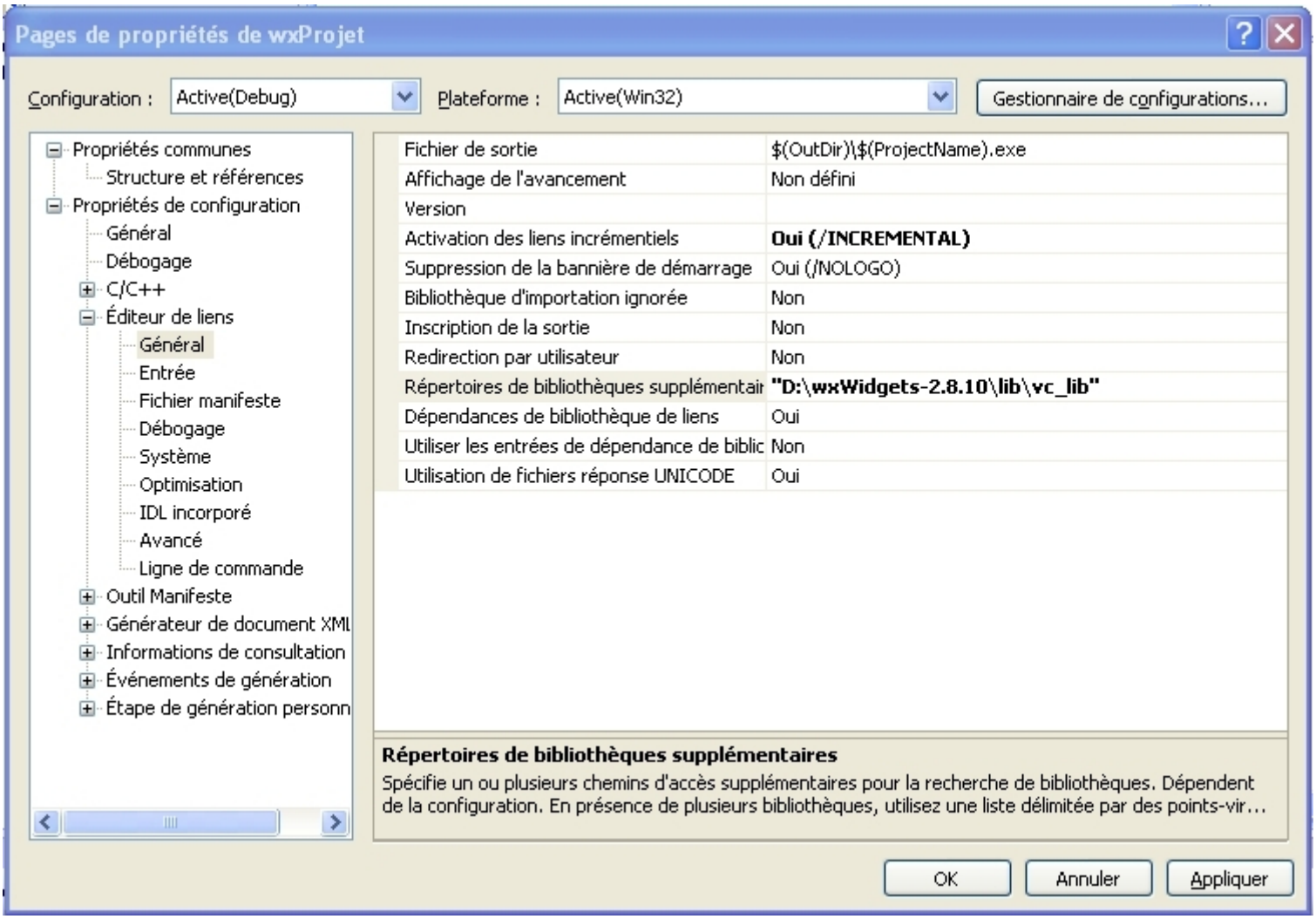
En cliquant sur le bouton avec les trois points '...', nous accédons à une boîte de dialogue où nous pouvons ajouter nos répertoires :



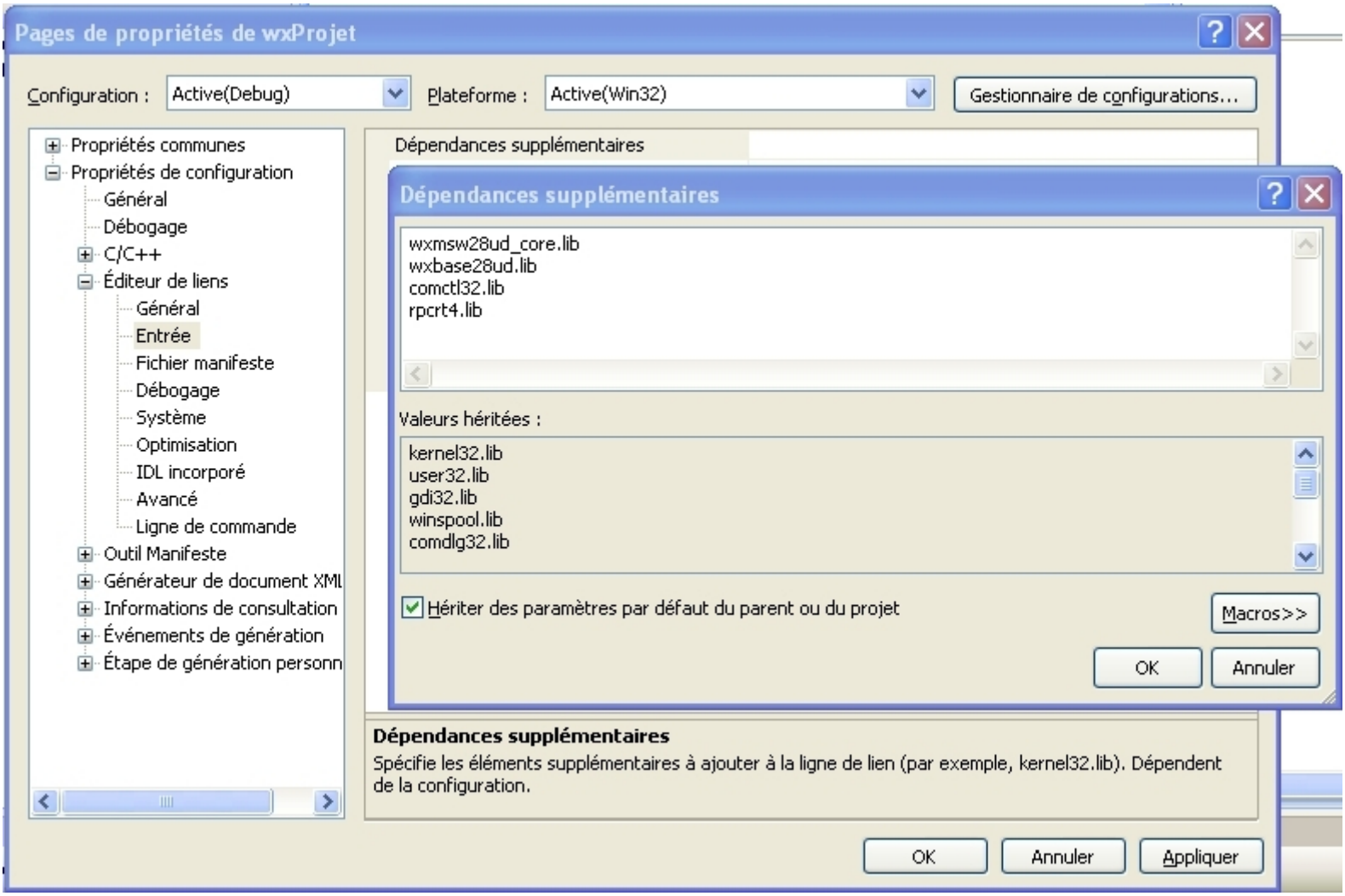
Attention à maintenir l'ordre d'inclusion : d'abord le répertoire de génération puis le répertoire de wxWidgets.
Pour le mode DEBUG, nous ajoutons la directive de compilation __WXDEBUG__ :



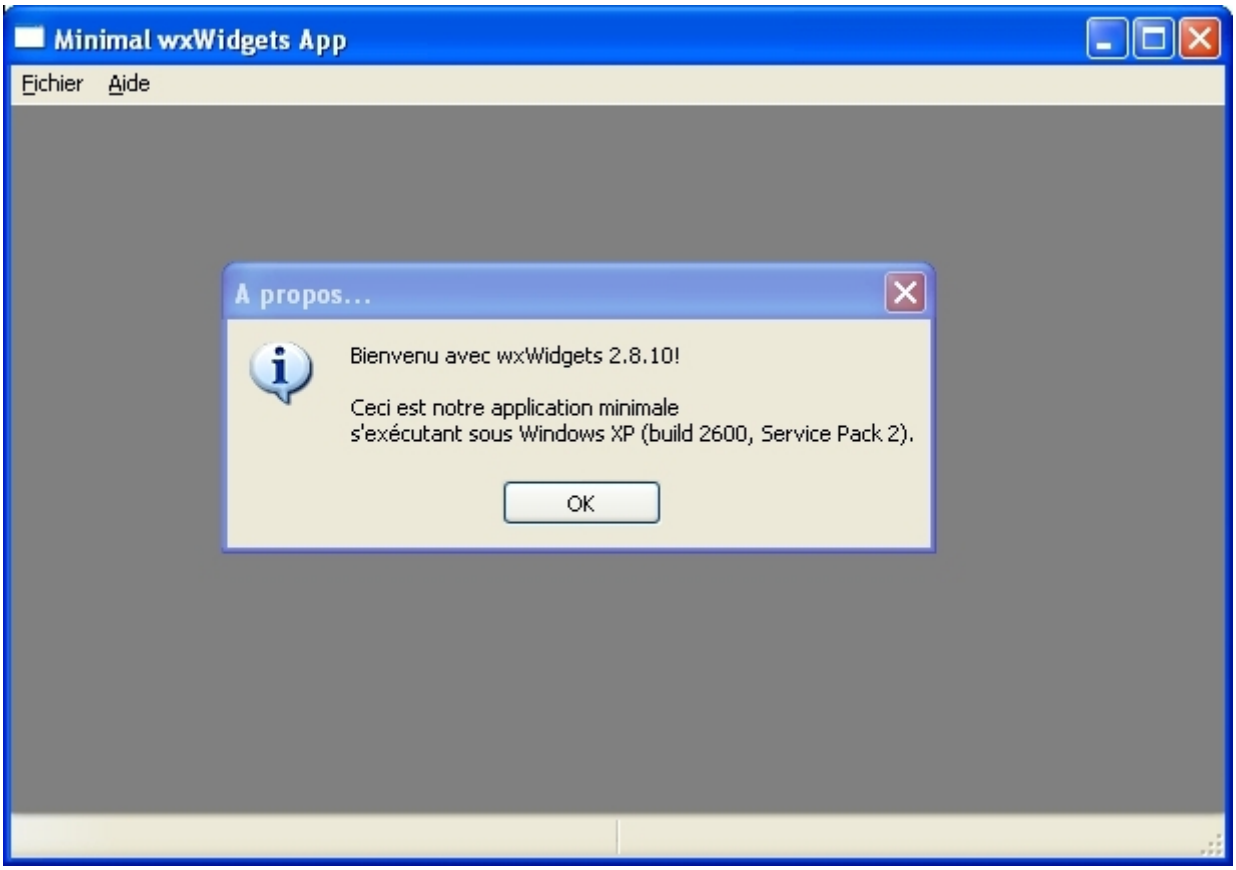
Il faut aussi configurer les options de liens. Nous retournons dans l'arbre de gauche et choisissons l'entrée 'Propriétés de configuration', 'Editeur de liens', 'Général'. Dans le tableau de droite, l'entrée à modifier est 'Répertoires de bibliothèques supplémentaires' dans lequel nous ajoutons le répertoire où se trouve notre génération :



Enfin, dernier paramètre, les bibliothèques avec lesquelles nous souhaitons lier l'application. Ceci est précisé en choisissant 'Entrée' dans l'arbre de gauche et en modifiant la première ligne du tableau de droite 'Dépendances supplémentaires' où nous pouvons ajouter les deux seules bibliothèques dont nous avons besoin (wxmsw28ud_core.lib et wxbase28ud.lib). Attention à rajouter les deux bibliothèques Microsoft (comctl32.lib et rpcrt4.lib) nécessaires à notre projet :



Nous pouvons demander la génération de la solution et lancer l'exécution de notre application. C'est avec plaisir que nous obtenons alors :



Vous pouvez trouver les sources de ce projet dans l'archive suivante : Source **Projet**

III-D - Compiler les fichiers exemples

Le répertoire "\wxWidgets-2.8.10\samples\" regroupe des projets d'exemples illustrant différentes classes de wxWidgets. Chaque répertoire suit le même principe. Il contient le ou les fichiers sources ainsi que le fichier de définition du projet .dsp. Générer des exemples devient chose facile.
Placez-vous dans le répertoire '\wxWidgets-2.8.10\samples\access', ouvrez 'access.dsp', acceptez la conversion, choisissez la configuration 'Unicode Debug' et demandez la génération. La compilation d'un fichier exemple entraîne la création d'un répertoire vc_mswud contenant l'exécutable généré.

Il est possible de générer tous les exemples en une seule fois. Il suffit de se positionner sur le répertoire "\wxWidgets-2.8.10\samples". Celui-ci contient un projet 'samples.dsw' pilotant la construction de tous les exemples. Le lecteur est maintenant familier de la démarche à suivre : accepter les conversions, choisir la configuration 'Unicode Debug' et demander la génération.
La version express n'étant pas disponible avec les MFCs, le projet 'mfctest' ne peut être généré.

III-E - Pour les lecteurs pressés : résumé

1. **Télécharger wxWidgets**
2. Désarchiver wxWidgets (répertoire destination sans espace)
3. Ouvrez le fichier "\wxWidgets-2.8.10\build\msw\wx.dsw" et acceptez toutes les conversions vers le nouveau format de projet
4. Modifier le setup : "\wxWidgets-2.8.10\include\wx\msw\setup.h" :

```
#define wxUSE_UNICODE 1
#define wxUSE_GLCANVAS       1
#define wxUSE_ODBC          1
#define wxUSE_STL 1
#define wxUSE_STD_IOSTREAM  1
```

5 Choisissez la configuration active 'Unicode Debug' et Lancez la génération

6 Paramétrer Visual C++
- 'Projet'/'Propriétés de wxProjet...' - Arbre de gauche : 'Propriétés de configuration', 'C/C++' - tableau de droite 'Autres répertoires Include' Ajout de "\wxWidgets-2.8.10\lib\vc_lib\mswud" et "\wxWidgets-2.8.10\include" dans cet ordre
- 'Projet'/'Propriétés de wxProjet...' - Arbre de gauche : 'Propriétés de configuration', 'C/C++', 'Préprocesseur' - tableau de droite 'Définitions du préprocesseur', ajoute '__WXDEBUG__'
- 'Projet'/'Propriétés de wxProjet...' - Arbre de gauche : 'Propriétés de configuration', 'Editeur de liens', 'Général' - tableau de droite 'Répertoire des bibliothèques supplémentaires' Ajout de "\wxWidgets-2.8.10\lib\vc_lib"
- 'Projet'/'Propriétés de wxProjet...' - Arbre de gauche : 'Propriétés de configuration', 'Editeur de liens', 'Entrée' - tableau de droite 'Dépendances supplémentaires' Ajout de wxmsw28ud_core.lib, wxbase28ud.lib, comctl32.lib et rpcrt4.lib

III-F - Et avec les autres versions ?

Normalement, cette démarche doit fonctionner avec les dernières versions payantes de Visual C++ 2008. Cependant, je n'ai pas testé cette compilation avec ces produits, donc je ne peux vous garantir à 100% qu'il n'y a pas nécessité de quelques ajustements.