Migrations de codes MFC de Visual 6.0 ou .net vers Visual studio 2005.
Par Farscape

1. Introduction

Depuis que je dispose de **Visual studio 2005** j'ai entrepris la migration de mes projets vers **Visual studio 2005.**
La transformation d'un projet **VC6** vers **VC2005** se déroule sans problèmes avec l'assistant.
Il est à noter que la prise en compte d'un projet sous **sourcesafe** ne cause pas problème tout est pris en charge correctement, un très bon point donc pour la migration d'un projet existant.
Les choses sérieuses commencent avec la compilation des sources.

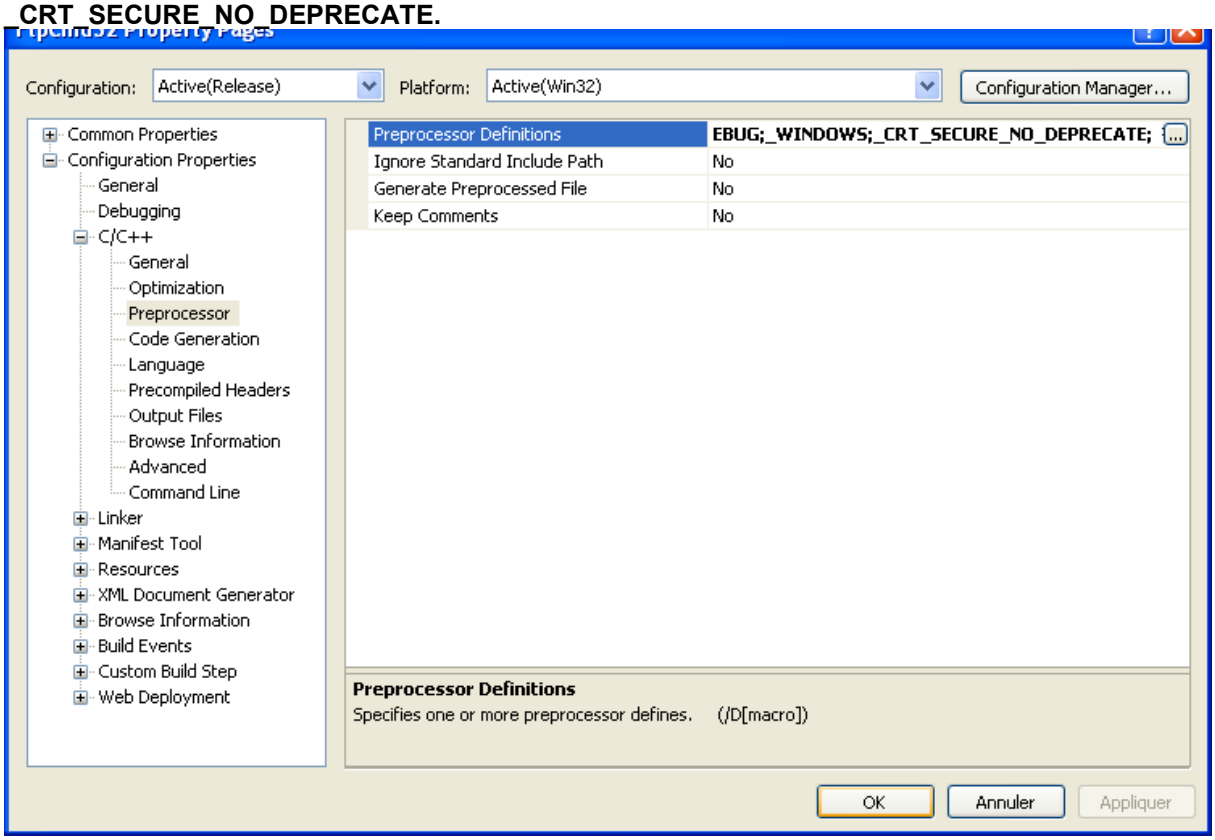
2. Erreurs de compilations :

C'était à prévoir on récolte pas mal d'erreurs et d'avertissements à la compilation même avec un projet issu de **Visual 2003**.

Voici une liste des plus courantes :

2.1 Fonctions liées à la bibliothèque CRT :

Warning:
C:\Program Files\Microsoft Visual Studio 8\VC\include\tchar.h(1467) : see declaration of '_tcsncpy'
Message: 'This function or variable may be unsafe. Consider using _tcsncpy_s instead. To disable deprecation, use _CRT_SECURE_NO_DEPRECATE. See online help for details.'
Explication:
Avec Visual C++ 2005 Microsoft a procédé à une profonde retouche de la bibliothèque **CRT** visant à améliorer la sécurité voir l'article MSDN:
Pour enlever ce warning on rajoutera dans les options de compilations la directive **_CRT_SECURE_NO_DEPRECATE.**



Ou on procédera à la transformation de code pour toutes les anciennes fonctions de la **CRT**…

2.2 Fonctions dépréciées:

Warning :
warning C4996: 'write' was declared deprecated
C:\Program Files\Microsoft Visual Studio 8\VC\include\io.h(334) : see declaration of 'write'
Les fonctions à la norme **POSIX** sont dépréciées avec Visual 2005.
En l'occurrence ici il faudra utiliser **_write** .
Il en sera de même pour **unlink ,rename** etc…

2.3 Erreurs liées au respect de la norme C++ :

2.3.1 Type Enuméré ou identifiant :

Warning :
c:\samples\valid\eval.h(95) : warning C4482: nonstandard extension used: enum ' TestType:: eType' used in qualified name
Le code provoquant le warning:

```
class TestType
{
//……
enum eType
        {
                TypeOne,
                TypeTwo
        };
void    SetMode(eType nModeType= eType::TypeOne){m_nMode=nModeType;}

eType m_nMode;

};
```

Explication : dans la classe l'identifiant est inutile:

```
// C4482.cpp
// compile with: /c /W1
struct S {
   enum E { a };
};

int i = S::E::a;   // C4482
int j = S::a;    // OK
```

2.3.2 Déclaration ou retour d'un entier par défaut :

La déclaration de variable ou de retour de fonction retournant un **int** par défaut n'est plus reconnue par Visual 2005.
Exemple:

```
const MAX=1000; //equivalent à  'const int MAX'
static n; // equivalent a 'static int n;'
func(char * p); // int func(char *p);
```

Pour la fonction **func** le compilateur retournera:

c:\samples\samplemdi\samplemdiview.h(13) : error C4430: missing type specifier - int assumed. Note: C++ does not support default-int
c:\samples\samplemdi\samplemdiview.h(13) : warning C4183: 'func': missing return type; assumed to be a member function returning 'int'

2.3.3 Portée d'une variable:

Ce code provoque une erreur :

```
for( int i=0; i<iLength; ++i )
     if( pszLine[i]==_T(';') )
        break;

  if( i < iLength)
  {
.....
```

Conformément à la norme C++ la portée d'une variable, c'est le bloc ou la boucle où elle est définie. Donc ici après la boucle **for** la variable n'existe plus.

Note : ce code était valide avec Visual C++6.0 et Visual .net 2003.
Solution : Le plus sage pour éviter de réfléchir sur la suite du code est de sortir la déclaration de la variable de la boucle.

```
int i=0;
for(i=0; i<iLength; ++i )
     if( pszLine[i]==_T(';') )
        break;
  if( i < iLength)
  {
.....
```

Cette erreur a représenté en ce qui me concerne mon plus gros travail de migration de code.

2.3.4 Déclaration d'un pointeur sur une fonction membre d'une classe :

```
class CMyClass
{
public:
 void function();
};
void (CMyClass::*pmf)() = CMyClass::function; //autorisé en visual C++ 6.0
```

Ce code était autorisé avec Visual C++ 6.0 et génère une erreur avec Visual C++2005 :

c:\samples\samplemdi\samplemdiview.cpp(114) : error C3867: 'CMyClass::function': function call missing argument list; use '&CMyClass::function' to create a pointer to member

Le code correct corrigé fonctionnant aussi avec visual C++ 6.0

```
class CMyClass
{
public:
 void function();
};
void (CMyClass::*pmf)() = &CMyClass::function;
```

2.3.5 Problèmes d'accès à des fichiers d'entêtes de définitions

Certains chemins de fichiers ont changé :

```
#if _MFC_VER < 0x0700
#include <..\src\afximpl.h>
#else
#include <..\src\mfc\afximpl.h>
#endif
```

Un grand classique avec la norme C++ sur la définition des entêtes :

```
#include <fstream.h>  // incorrect ,à remplacer par  <fstream>
```

En effet depuis la normalisation ISO du C++ en 1998 les entêtes standards n'ont plus l'extension .h et font partie de l'espace de nommage standard **std.**
Il faudra donc veiller à ne pas oublier de rajouter dans vos sources la ligne :

```
using namespace std;
```

ou d'utiliser explicitement **std ::**
Exemple :

```
std::ifstream fichier( "fichier.txt" );
```

Visual C++ 6 date aussi de 1998 mais autorise les deux formes.

2.4 Erreurs liées aux classes MFC:

La gestion des exceptions avec la classe **CException** génère quelques erreurs notamment :
CFileException::generic
Qu'il faudra remplacer par :

```
#if _MFC_VER >= 0x0800
case CFileException::genericException:;
#else
case CFileException::generic:;
#endif
```

Autre exemple de problème avec la classe **CFile** (depuis .net2001) :

```
BOOL CDIBPal::Save(CFile* fp)
 {
#if _MFC_VER >= 0x0700
   return Save((UINT)fp->m_hFile);
#else
   return Save(fp->m_hFile);
#endif
}
```

2.4.1 Quelques définitions de messages dont les signatures ont changé

Note : les solutions proposées ci-dessous permettent avec l'utilisation des **defines** d'avoir un code compatible avec les différentes versions du compilateur : de **Visual 6.0** à **Visual studio 2005.**

Avec Visual 2005:

```
#if _MFC_VER >= 0x0800
afx_msg LRESULT OnNcHitTest(CPoint point);
#else
afx_msg UINT OnNcHitTest(CPoint point);
#endif
```

C'est un exemple il y en a certainement d'autres.

Astuce :
Il suffit de regarder le message d'erreur généré, ou bien à l'aide d'un projet bidon de test réalisé en Visual 2005 de générer le message qui cause un problème pour voir comment il est implémenté et d'appliquer la modification dans le projet.

A partir de Visual 2003 :

```
#if _MFC_VER >= 0x0700
LRESULT CMyButton::OnDrawItem(WPARAM wParam, LPARAM lParam)
{
   int nIDCtl=wParam;
   LPDRAWITEMSTRUCT lpDrawItemStruct=(LPDRAWITEMSTRUCT)lParam;
#else
void CMyButton::OnDrawItem(int nIDCtl,LPDRAWITEMSTRUCT lpDrawItemStruct)
{
#endif

#if _MFC_VER >= 0x0700
return 0L;
#endif
}
```

Dans le .h section Afx :

```
#if _MFC_VER >= 0x0700
LRESULT OnDrawItem(WPARAM wParam, LPARAM lParam);
#else
afx_msg void OnDrawItem(int nIDCtl, LPDRAWITEMSTRUCT lpDrawItemStruct);
#endif
```

Certaines structures sont différentes, exemple:

```
#if _MFC_VER >= 0x0700
struct CCharFormat : public CHARFORMAT2
#else
struct CCharFormat : public CHARFORMAT
#endif
```

3. Problèmes rencontrés à l'exécution du programme :

3.1 Cohabition de compilateurs Microsoft de version différente :

Un conseil si vous utilisez deux compilateurs sur votre machine par exemple **Visual 6.0** et **Visual studio 2005**
Veillez à bien paramétrer dans vos projets des répertoires distincts pour les deux compilateurs pour les fichiers de sortie (.obj ,.lib) .

Donc faites attention à ne pas mélanger les .lib générés en debug.
Sinon vous risqueriez d'avoir des messages vous indiquant qu'il manque une dll pour l'exécution du programme par exemple **msvcrt80.dll**.

3.2 La bibliothèque CRT standard :

Comme indiqué plus haut la bibliothèque CRT a été profondément remaniée si bien que dans le mode multithread DLL c'est la dll **msvcrt80.dll** qui sera utilisée au lieu de **msvcrt.dll**,
Dans ce contexte il est préférable (si possible) de reconstruire toutes les bibliothèques utilisées par votre programme.
On pourra se reporter à ce lien MSDN pour plus de renseignements :
http://msdn2.microsoft.com/en-us/library/abx4dbyh.aspx

Et plus particulièrement le paragraphe: **What problems exist if an application uses both msvcrt.dll and msvcr80.dll?**

3.3 Plantage en fin d'exécution:

Sur tous mes projets transférés j'ai eu droit à un plantage en mode release en sortie de programme ;
L'assertion d'erreur indiquait un problème en rapport avec la classe **CImageList.**
Après avoir galéré pas mal de temps sur cette erreur, j'ai eu l'idée d'un problème sur l'alignement mémoire des structures.

Vos projets doivent respecter l'alignement par défaut des MFC ,il faudra donc utiliser des directives **pragma pack(push,1) (par exemple)** et **#pragma pack(pop)** pour vos structures réclamant un alignement particulier plutôt que de spécifier l'alignement dans les options de compilations.

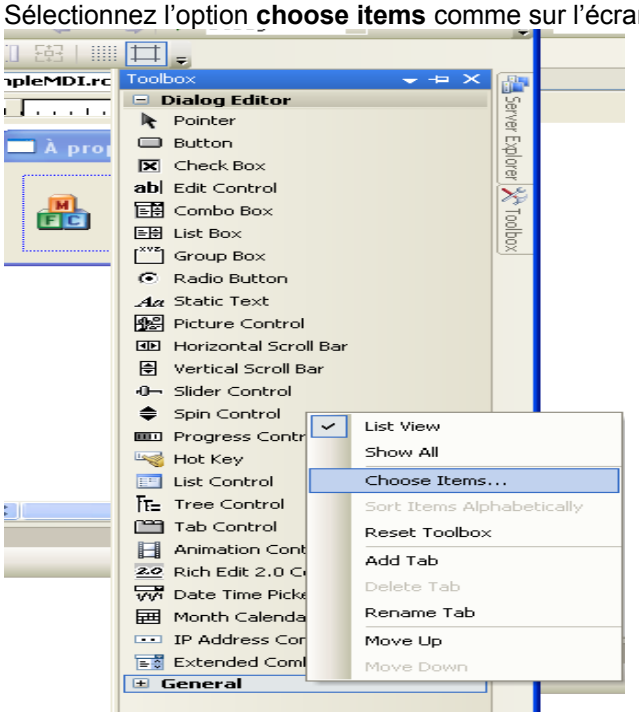
Ce problème est confirmé sur les forums MSDN :
http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=169231&SiteID=1

4. Utilisation et paramétrages des Activex dans l'environnement de développement :

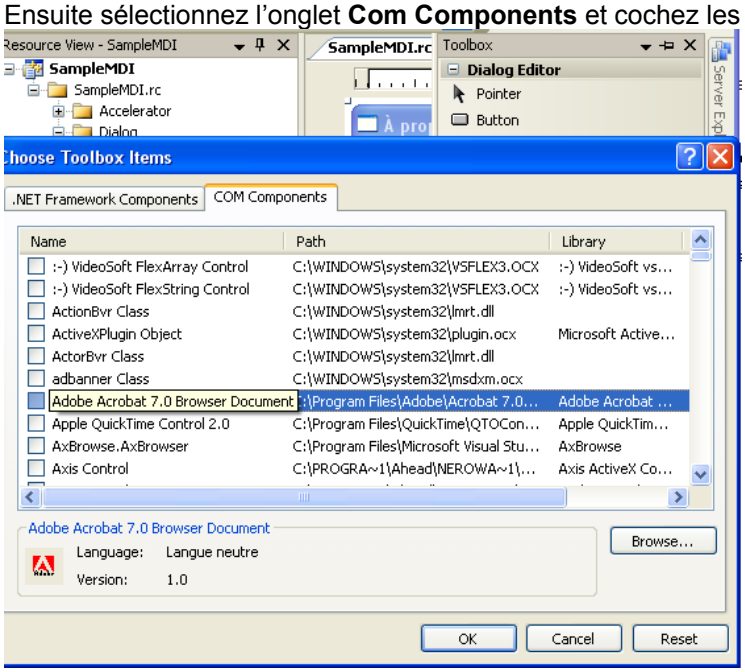
Avec **Visual 6.0** l'insertion d'un Activex dans un projet rajoutait dans la barre d'outils des ressources le composant et générait les classes d'interface (Wrappers) .
Avec **Visual 2005** les choses ne se passent pas tout à fait pareil :
L'insertion de vos Activex préférés est indépendante des projets, c'est-à-dire une fois renseignée dans la barre d'outils ils sont disponibles pour tous vos projets.
La génération des classes d'interface de l'Activex sera effectuée à part.
Comment procéder ?

4.1 Insertion des Activex :

Sélectionnez l'option **choose items** comme sur l'écran ci-dessous :



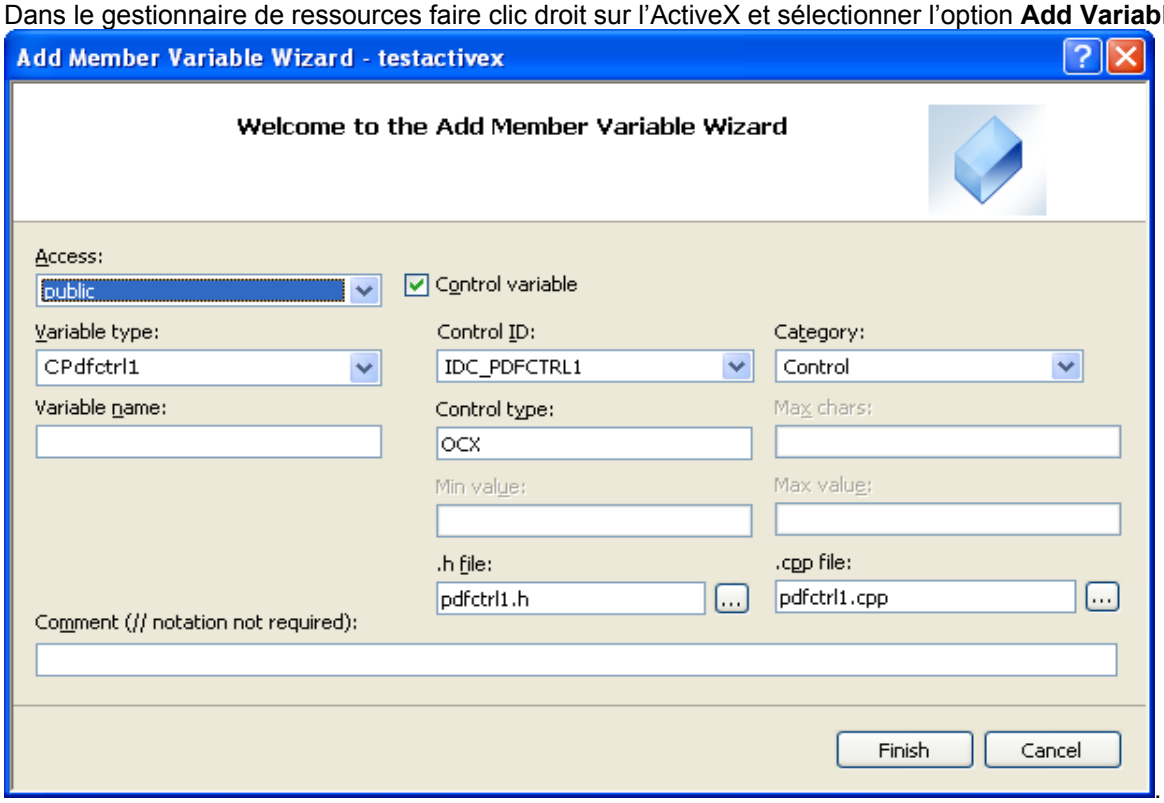
Ensuite sélectionnez l'onglet **Com Components** et cochez les Activex à insérer :



4.2 Générations des classes d'interface (Wrappers) :

La génération des classes d'interfaces se fera lors de l'association d'une variable contrôle à l'Activex dans les ressources.

Dans le gestionnaire de ressources faire clic droit sur l'ActiveX et sélectionner l'option **Add Variables**



Les classes d'interfaces sont alors générées dans le projet avec les noms pré remplis dans les champs « **.h File : »** et **« .cpp File. »**

4.3 Une petite information complémentaire :

Le fichier **.vcproj** de votre projet contient en fin de fichier les définitions de vos Activex et la classe par défaut générée lors de l'ajout d'une variable contrôle sur l'Activex.

Si ces définitions ne sont pas reprises dans votre projet vous pouvez les rajouter manuellement. Pour cela ouvrez avec Notepad votre ancien projet VC6 le fichier **.dsp** à la fin du fichier vous trouverez l'identifiant de l'Activex et sa classe pour la génération du contrôle.

Il vous suffira de rajouter dans le fichier **.vcproj** en fin de fichier les définitions correspondantes sous la forme :

```
……
</Files>
	<Globals>
		<Global
			Name="{183504C5-44F4-11D1-AE90-703348C10001}"
			Value="CMyCtrl"
		/>
		<Global
			Name="{5B109B4EC3-CDE9-11CF-9A3F-0190C81874C8}"
			Value="CMyCtrl2"
		/>
	</Globals>
</VisualStudioProject>
```

Name correspond à l'identifiant de l'Activex et **Value** le nom de la classe par défaut générée (on peut donner un autre nom).

5. Distribution d'un programme MFC :

Pour distribuer les dll des MFC sur un poste nous disposons d'un setup de distribution **vcredist_x86.exe** disponible à l'emplacement :
C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\BootStrapper\Packages\vcredist_x86\vcredist_x86.exe

6. Le Gestionnaire de classes (ClassView)

Avec Visual C++ 2005 le ClassView est dynamique et repose sur le fichier **.ncb**, il faudra penser à supprimer ce fichier issu de votre ancienne configuration pour que Visual 2005 le reconstruise.

Je pense être tombé sur un bug du ClassView, ce problème est lié à l'utilisation de classes MFC dialogues (CFormView ,CDialog ,CDialogBar) construites à partir de classe patron (Template). Dans ce contexte la demande de génération d'une variable contrôle à partir des ressources échoue, l'option contrôle variable étant grisée.

Exemple :

```
template <class TPL_FORM=CFormView>
class  TTplForm : public TPL_FORM
{
public:

      TTplForm(UINT nIDTemplate):TPL_FORM(nIDTemplate){}
      ~TTplForm(){}

      void Fonction(){}
};


class CTestFormView : public TTplForm<CFormView>
{
protected: // create from serialization only
      CTestFormView();
      DECLARE_DYNCREATE(CTestFormView)
//....
```

Cette simple définition empêche l'insertion d'une variable contrôle à partir des ressources ce qui n'était pas le cas bien sûr avec Visual 6.0.

J'ai soumis le bug au support technique de Microsoft.

7. Conclusions :

J'ai évoqué les problèmes usuels il y a fort à parier que des problèmes de compilation surviennent sur une utilisation avancée des templates ou de la STL.

Voilà, j'espère que ce tour d'horizon vous permettra d'avancer dans la migration de vos projets.

Remerciements :
Je remercie toute l'équipe du forum Visual C++ pour sa relecture attentive du document

Je remercie également Anomaly pour la correction orthographique.