Création d'un ActiveX MFC avec Visual 2005 ou Visual 6.0

Par **Farscape**

Date de publication : Septembre 2006

Dernière mise à jour : 17 Septembre 2006

Cet article vous apprendra à créer des composants ActiveX avec les plateformes de développement **Visual studio 2005** et **Visual studio 6.0**

I– A. Qu'est ce qu'un ActiveX ?

Le site de Microsoft France fournit une définition simple :
Les contrôles ActiveX, autrefois connus sous le nom de contrôles OLE ou de contrôles OCX, sont des composants (ou objets) que vous pouvez insérer dans une page Web ou autre programme de manière à pouvoir réutiliser une fonctionnalité empaquetée programmée par quelqu'un d'autre.

http://support.microsoft.com/default.aspx?scid=kb;fr;154544

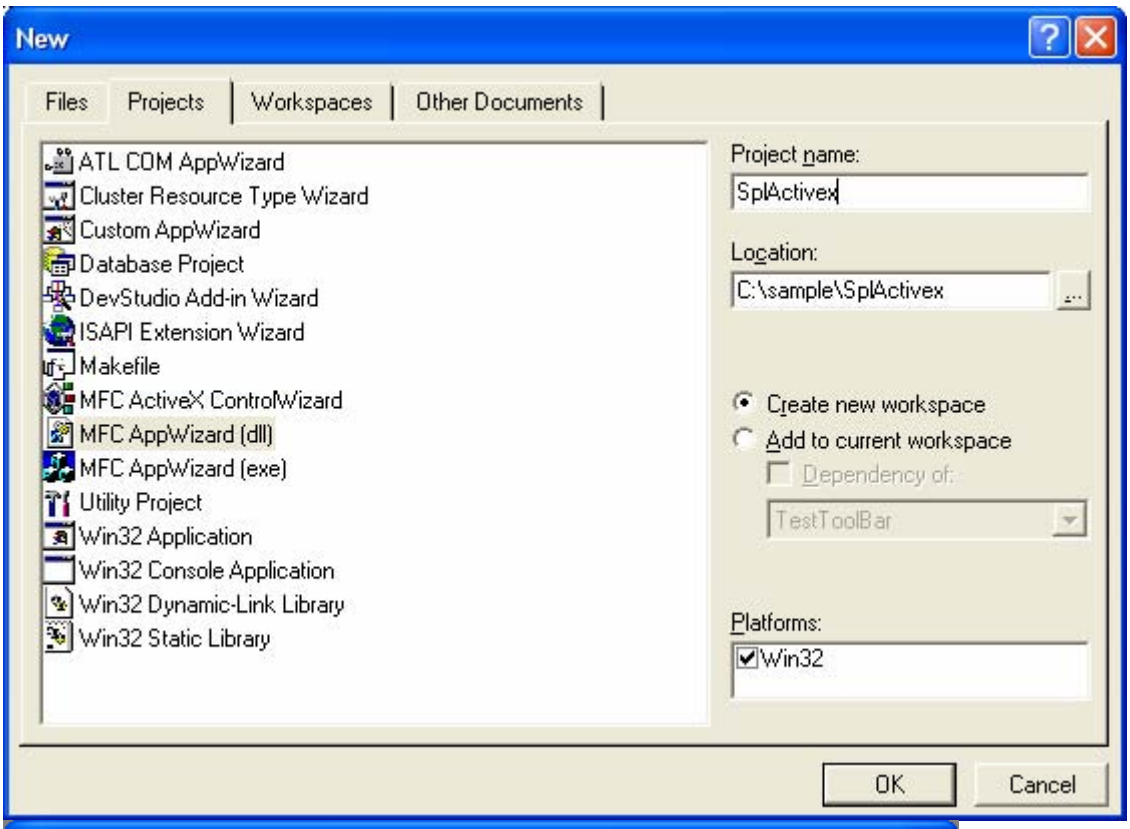
On peut créer un ActiveX en utilisant la bibliothèque MFC ou ATL .
Un ActiveX MFC impose d'avoir les DLL MFC présentes et à jour sur les postes cibles.
Ce qui n'était plus un problème depuis quelques années avec Visual 6.0 puisque l'on peut dire que l'ensemble des PC de dernière génération à base de Windows XP ou 2000 ou 2003 était à jour.
Ce n'est plus vrai avec Visual 2005, le nom des DLL ayant changé il faudra faire attention à la distribution.
Les contrôles ActiveX ATL s'affranchissent des MFC et sont de fait plus petits.

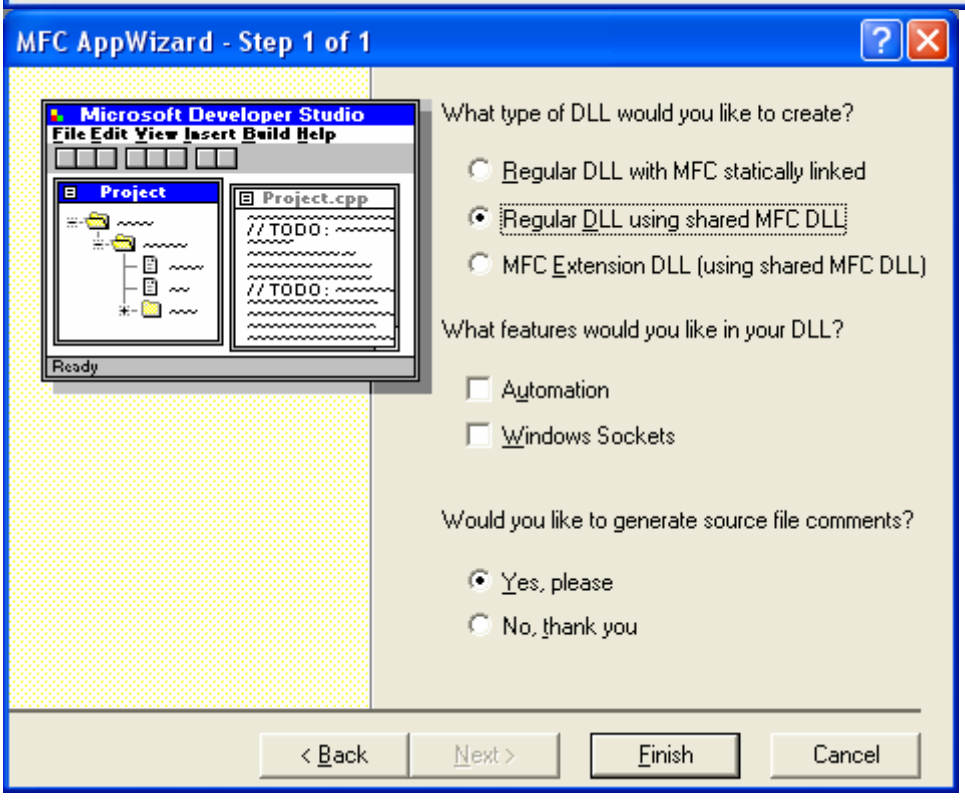
Dans cet article vous apprendrez à créer le squelette d'un ActiveX en utilisant les MFC.
Je vous présenterai dans un premier temps les étapes concernant la génération d'un ActiveX, l'ajout de méthodes, de propriétés et d'événements.
Puis nous verrons comment sauvegarder les propriétés de l'ActiveX.
Et enfin son intégration dans un programme MFC.

Dans un deuxième volet à venir, je mettrai tout ça en pratique avec la réalisation d'un bouton ActiveX

II . Génération de l'ActiveX avec l'assistant

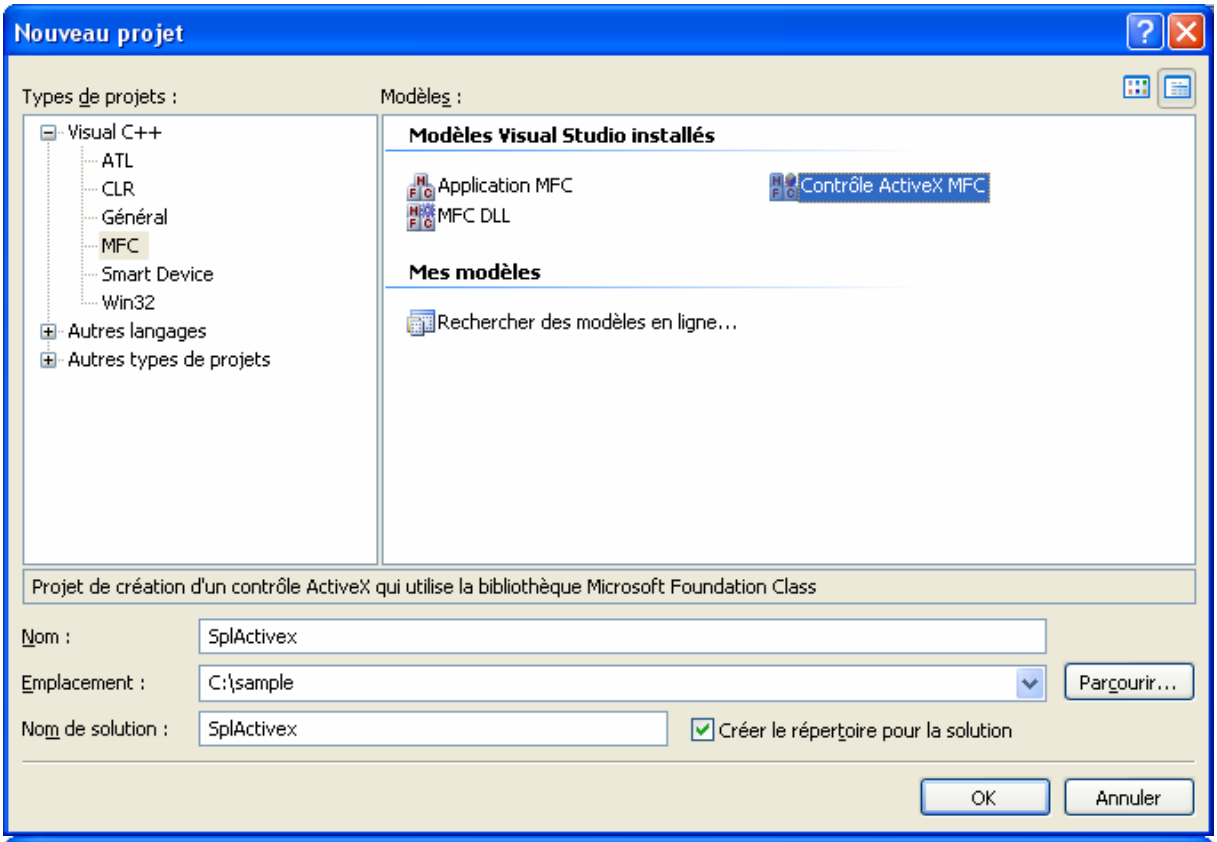
II-A. Avec Visual 6.0 :

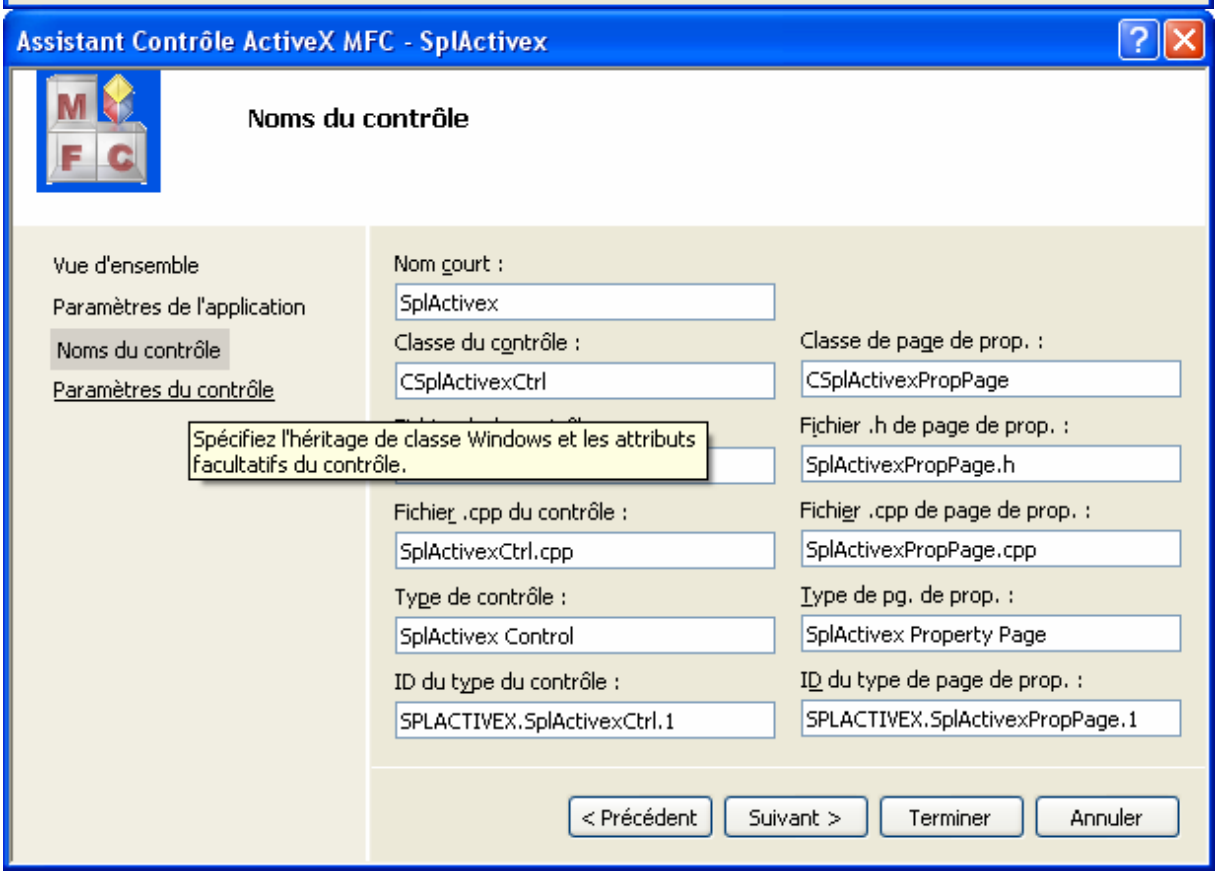


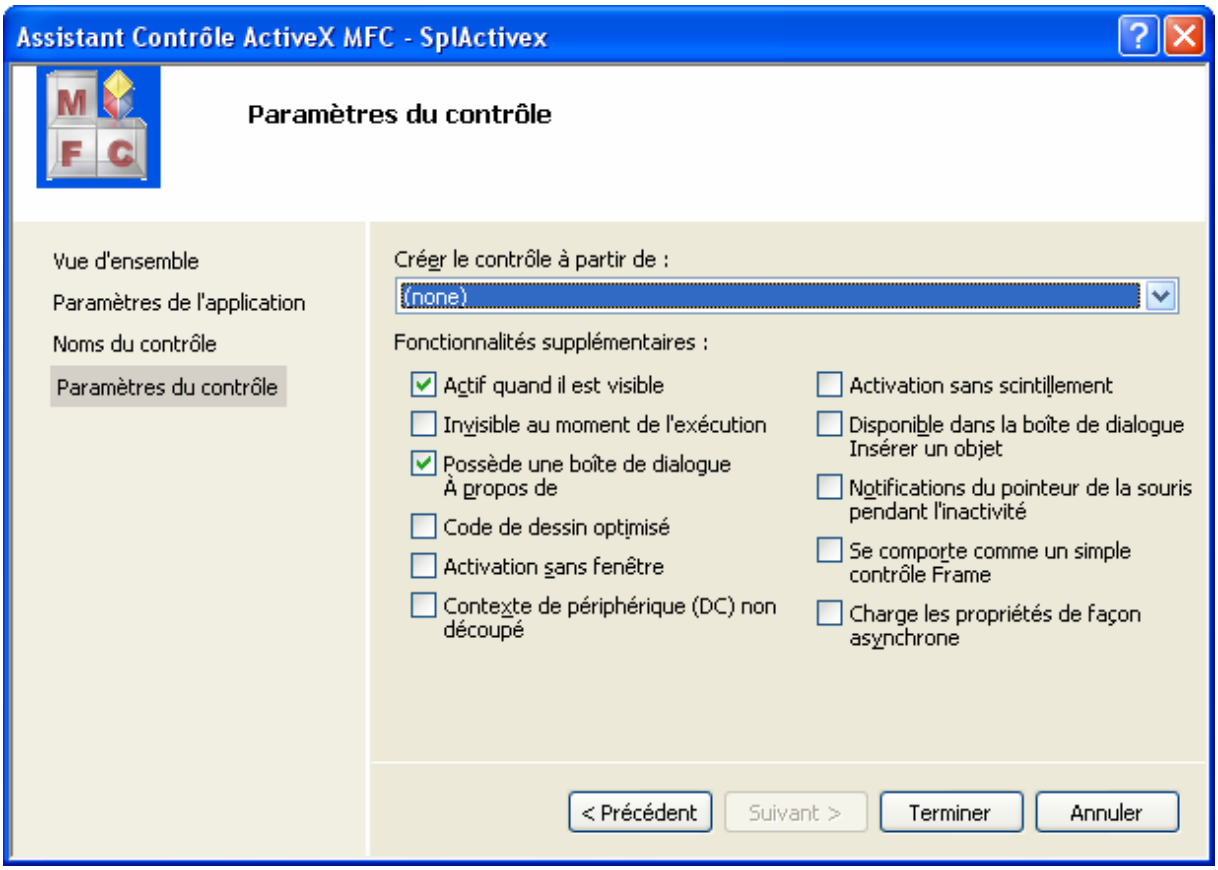


II-B. Avec Visual 2005 :

Les étapes de génération sont très simples :







L'option « créer le contrôle à partir de » est importante :
Si vous créez un nouveau contrôle qui n'a rien de commun avec les contrôles Windows existants, laissez le sélecteur à « none ».
Si vous développez un bouton personnalisé vous avez intérêt à sélectionner le type bouton pour votre contrôle.
Vous hériterez ainsi des styles propres au contrôle de base.

Ce choix est important si vous laissez le sélecteur à « none » le dessin et le comportement de l'ActiveX sera entièrement à votre charge.
En héritant d'un contrôle de base vous aurez donc la possibilité d'enrichir le fonctionnement d'un contrôle de base existant.
Les principaux contrôles étant :

- Le bouton
- La comboBox
- L'édit
- Le static
- La Listbox

II-C. Aperçu du code généré

Le source principal de l'ActiveX sera celui portant le nom du projet plus le suffixe **Ctrl** .
Dans mon exemple **SplActiveXCtrl**
La classe générée hérite de la classe **COleControl**.

II-C-1 . Les tables d'événements ou messages

Dans un ActiveX quatre types de table sont implémentés,

Une table d'événements :

```
BEGIN_EVENT_MAP(CSplActiveXCtrl, COleControl)
END_EVENT_MAP()
```

Cette table correspond à la gestion des événements souris ou claviers sur l'ActiveX .

Une table des propriétés ou table de dispatch :

```
BEGIN_DISPATCH_MAP(CSplActiveXCtrl, COleControl)
DISP_FUNCTION_ID(CSplActiveXCtrl, "AboutBox", DISPID_ABOUTBOX, AboutBox, VT_EMPTY, VTS_NONE)
DISP_PROPERTY_NOTIFY_ID(CSplActiveXCtrl, "MyVar", dispidMyVar, m_MyVar, OnMyVarChanged, VT_I2)
DISP_PROPERTY_NOTIFY_ID(CSplActiveXCtrl, "MyVarTwo", dispidMyVarTwo, m_MyVarTwo,
OnMyWarTwoChanged, VT_I4)
END_DISPATCH_MAP()
```

Cette table permet de faire le lien entre les propriétés définies et l'interface (classe wrapper) de l'ActiveX.
Quand une propriété (fonction Set, voir rubrique suivante) est modifiée la fonction associée est appelée dans l'ActiveX.

Une table des messages :

```
BEGIN_MESSAGE_MAP(CSplActiveXCtrl, COleControl)
      ON_OLEVERB(AFX_IDS_VERB_EDIT, OnEdit)
      ON_OLEVERB(AFX_IDS_VERB_PROPERTIES, OnProperties)
END_MESSAGE_MAP()
```

Une table pages de propriétés :

```
// TODO : ajoutez autant de pages de propriétés que nécessaire. N'oubliez
pas d'augmenter le compteur !
BEGIN_PROPPAGEIDS(CSplActiveXCtrl, 1)
      PROPPAGEID(CSplActiveXPropPage::guid)
END_PROPPAGEIDS(CSplActiveXCtrl)
```

Cette table gère la liste des pages de propriétés de votre ActiveX.
C'est l'équivalent des onglets de propriétés d'un contrôle MFC dans l'éditeur de ressources.

Si votre interface doit comporter plus d'une page (celle générée par défaut), il faudra venir rajouter manuellement sa définition dans cette table.

La page de propriétés permettra donc de définir une interface graphique pour modifier les propriétés de l'ActiveX.

II-C-2 . Dessin de l'ActiveX :

Le dessin de l'ActiveX doit être effectué dans la fonction **OnDraw** :

```
// CSplActiveXCtrl::OnDraw - Fonction de dessin

void CSplActiveXCtrl::OnDraw(
                    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
      if (!pdc)
            return;

      // TODO : remplacez le code suivant par votre code dessin.
      pdc->FillRect(rcBounds,
CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
      pdc->Ellipse(rcBounds);

      if (!IsOptimizedDraw())
      {
      // Le conteneur ne prend pas en charge le dessin optimisé.
// TODO : si vous avez sélectionné un objet GDI quelconque dans le contexte de
périphérique *pdc,
            //restaurez ici les objets précédemment sélectionnés.
      }
}
```

L'assistant génère par défaut le dessin d'une ellipse.
Votre logique de dessin devra donc être écrite à cet emplacement et devra tenir compte des différentes propriétés de l'ActiveX.

III. Ajout d'une propriété dans un ActiveX

III-A . Qu'est ce qu'une propriété dans un ActiveX ?

L'ActiveX gère deux types de propriétés :

III-A-1. Les propriétés définies par l'utilisateur (custom)

Une propriété sera une donnée membre avec une interface d'accès en lecture et écriture.
En clair une fois définie, le wrapper de classe de l'ActiveX disposera d'une fonction **Get** et **Set** correspondant à cette variable.

Dans la classe contrôle de l'ActiveX une fonction de notification est générée pour gérer le changement de la variable.
L'utilisateur est libre de compléter son traitement et de provoquer par exemple le rafraîchissement du composant.
Enfin la sauvegarde et la lecture de ces données personnalisées devra être faite dans une fonction dédiée.

III-A-2. Les propriétés du stock

L'ActiveX possède aussi une liste de propriétés disponibles que l'on nomme propriétés du stock, celles-ci sont définies dans la classe **COleControl**

propriété	Entrée dans la table	Accès à la valeur par …
Appearance	**DISP_STOCKPROP_APPEARANCE( )**	La variable **m_sAppearance**
BackColor	**DISP_STOCKPROP_BACKCOLOR( )**	L'appel de **GetBackColor()**
BorderStyle	**DISP_STOCKPROP_BORDERSTYLE( )**	La variable **m_sBorderStyle**
Caption	**DISP_STOCKPROP_CAPTION( )**	L'appel de **InternalGetText()**
Enabled	**DISP_STOCKPROP_ENABLED( )**	La variable **m_bEnabled**
Font	**DISP_STOCKPROP_FONT( )**	Le DC est affecté avec **SelectStockFont**
ForeColor	**DISP_STOCKPROP_FORECOLOR( )**	L'appel de **GetForeColor()**
hWnd	**DISP_STOCKPROP_HWND( )**	La variable **m_hWnd**
Text	**DISP_STOCKPROP_TEXT( )**	L'appel de **InternalGetText()** identique à Caption
ReadyState	**DISP_STOCKPROP_READYSTATE()**	m_lReadyState ou **GetReadyState()**

Les propriétés de stock correspondent à des fonctionnalités prévues pour l'ActiveX un peu comme des options que l'on peut rajouter selon un catalogue …

Ces options sont en relations avec le traitement de l'apparence de l'ActiveX, exemple :
BackColor et **Forecolor** devra correspondre dans votre traitement à la couleur de fond et à la couleur d'écriture dans votre ActiveX.

Autres points:

- Les propriétés du stock sont sauvegardées et lues automatiquement par le container.
- Les propriétés du stock ne génèrent pas de fonctions **Get** et **Set,** leurs modifications entraînent un rafraîchissement automatique du contrôle.

III-A-3. Une propriété pour quoi faire ?

La Notion de propriété correspond à celles des contrôles Windows existants, exemple la notion lecture seule sur un Edit.

Lorsque l'utilisateur utilisera une méthode pour changer la valeur d'une propriété custom de l'ActiveX, l'action se terminera dans l'ActiveX par un message type **OnxxxChanged** :

```
void CSplActiveXCtrl::OnMyVarChanged(void)
{
	AFX_MANAGE_STATE(AfxGetStaticModuleState());

	// TODO : ajoutez ici le code de votre gestionnaire de propriété
	MessageBox(_T("MyVarChanged"),_T("SplActiveXCtrl"));
	SetModifiedFlag();
}
```

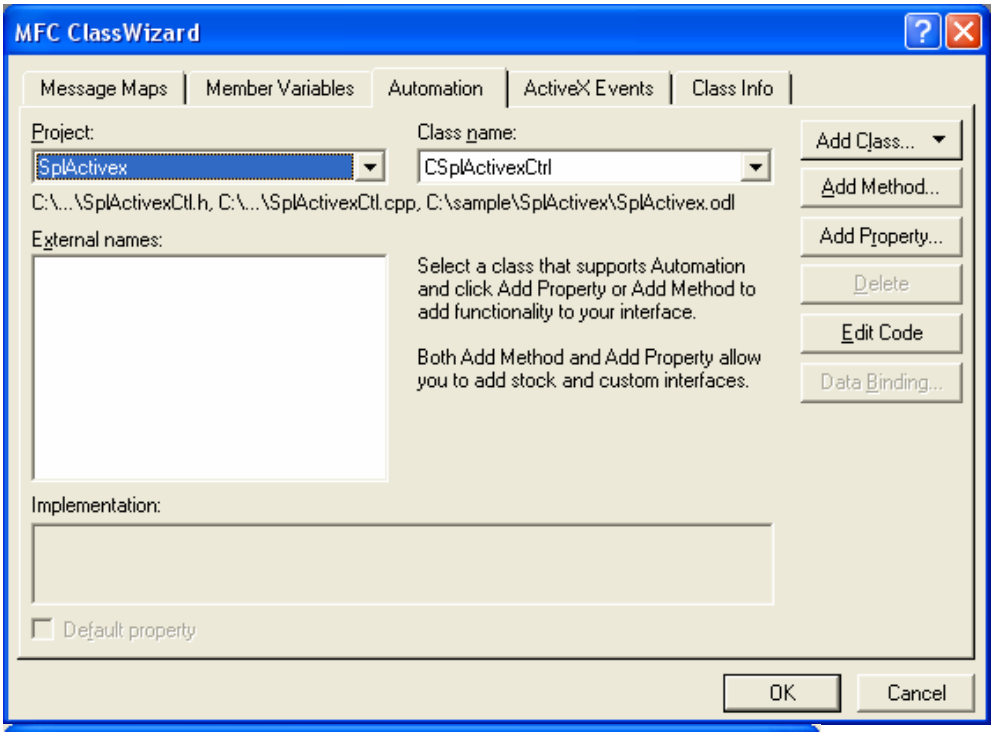
Ici la propriété se nomme **MyVar** à laquelle correspond donc un message de notification de changement de valeur dans l'ActiveX.

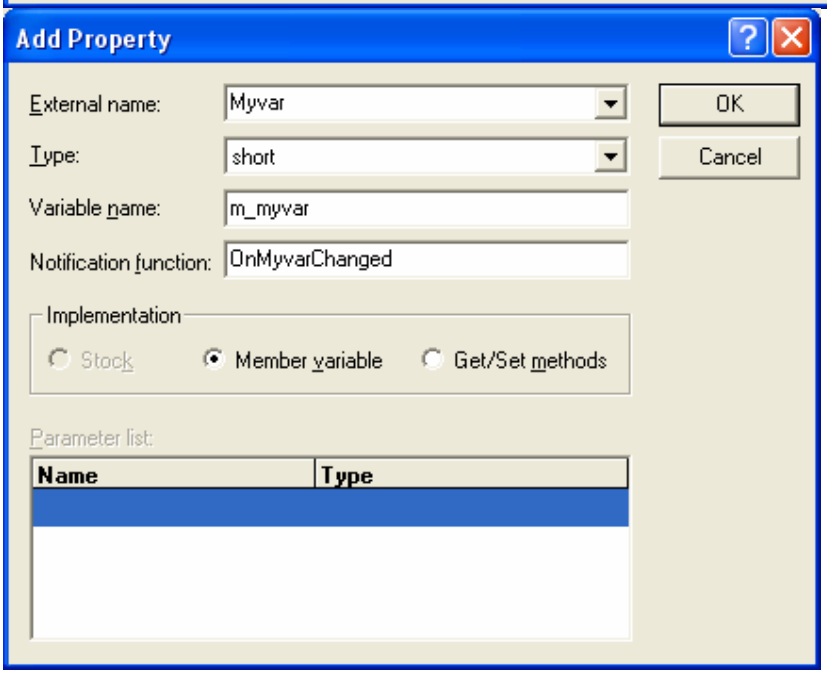
Il reste à prendre en charge l'éventuel changement d'apparence du contrôle en provoquant par exemple un rafraichissement du dessin.
Pour finir le nom des propriétés que vous allez définir sera visible dans les propriétés de l'ActiveX à partir de l'éditeur des ressources ; il est donc capital **de donner du sens** au nom de vos variables …

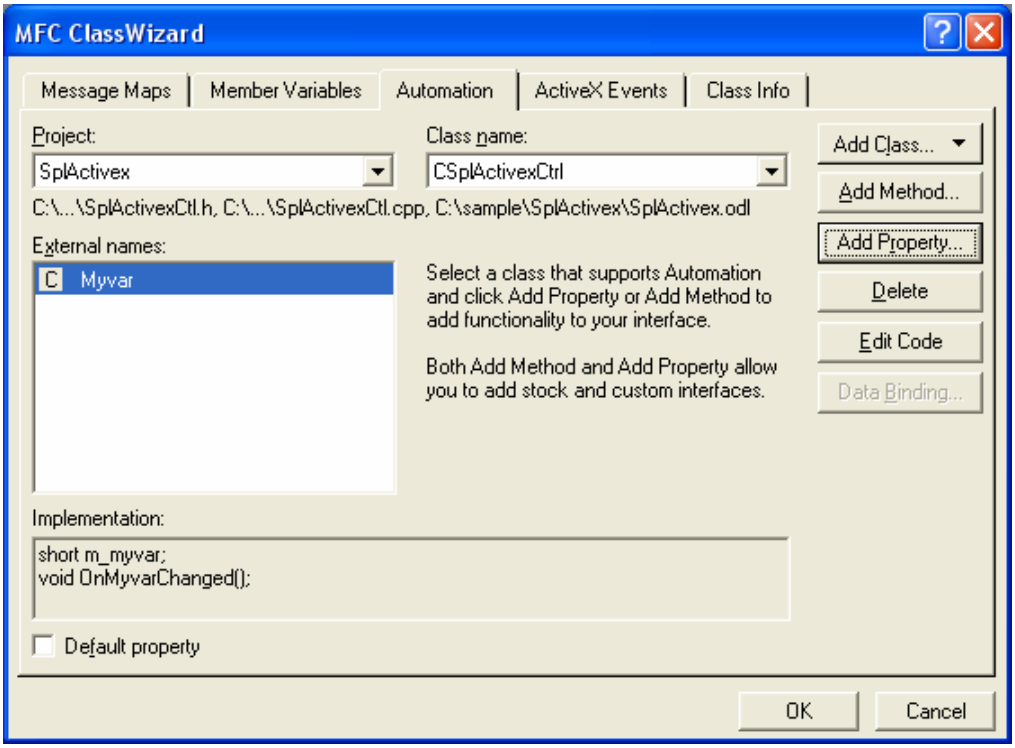
Pour rajouter une propriété sur un ActiveX on utilisera l'assistant qui varie selon les versions de Visual.

III-B. Avec Visual 6.0 :

Appelez **ClassWizard** par le menu ou le raccourci **CTRL+W**
Ensuite sélectionnez l'onglet automation, puis cliquez ensuite sur le bouton **Add Property**.
Dans le sélecteur **external name** on dispose des noms de propriétés disponibles pour l'ActiveX (stock).
Si c'est une propriété différente de celles du stock tapez le nom de votre variable et sélectionnez ensuite le type approprié dans le sélecteur de Type.
Enfin cliquez sur le bouton OK.

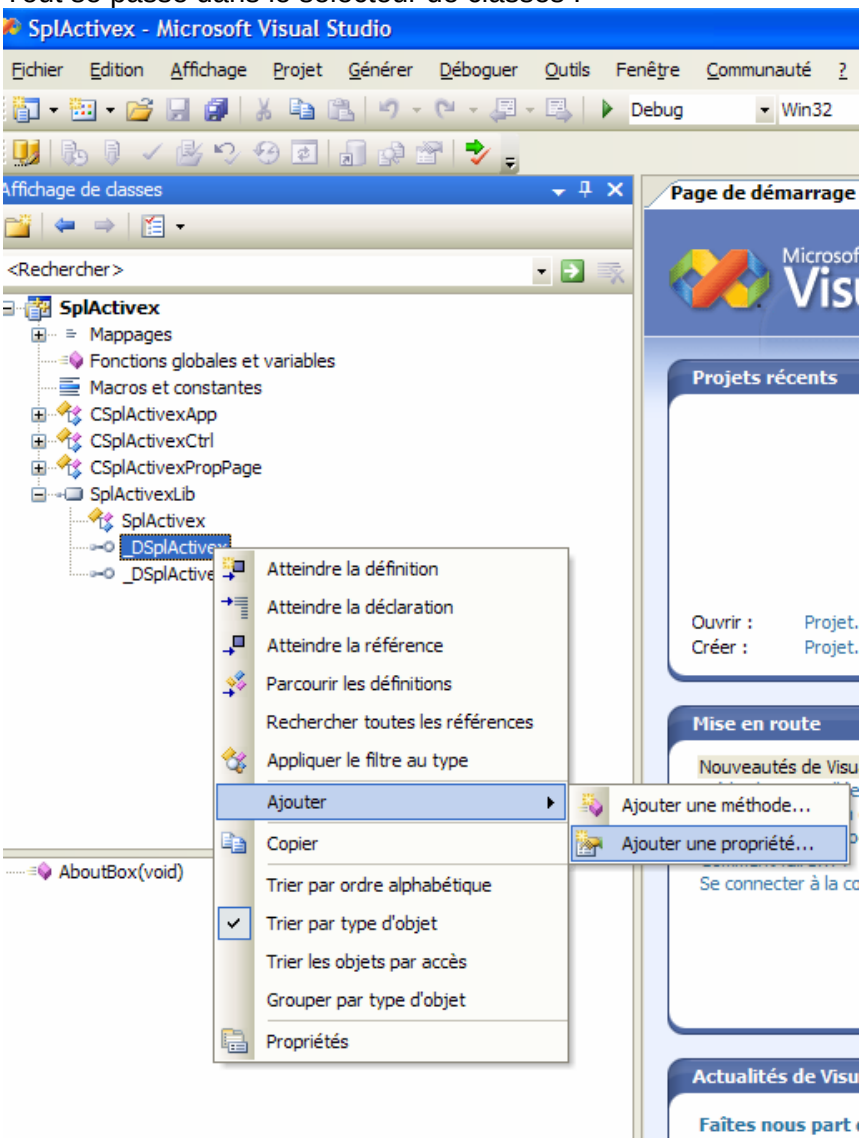


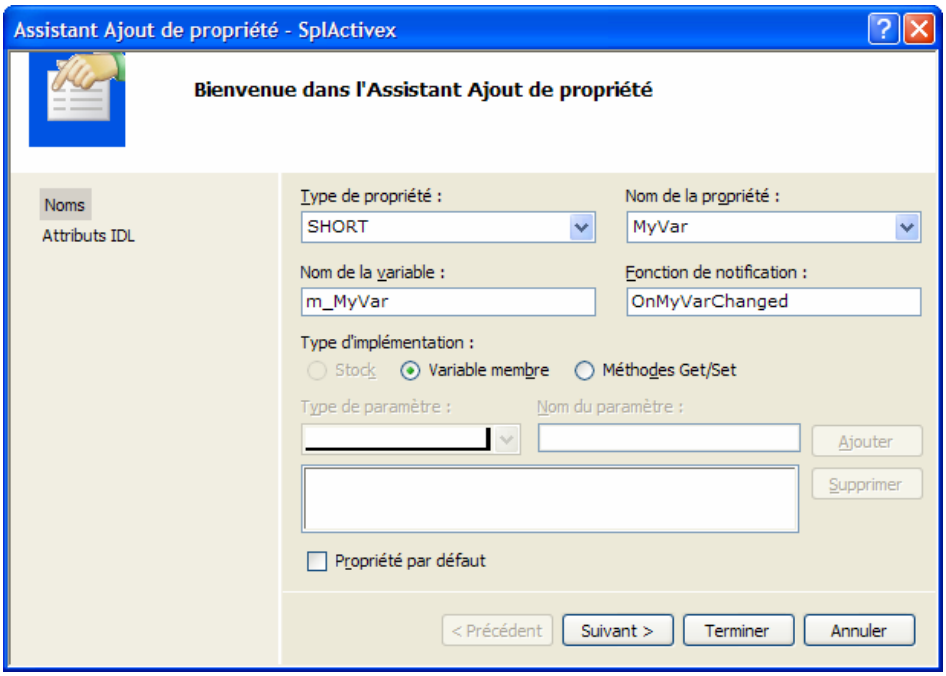



MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
SplActivex
Class name:
CSplActivexCtrl
Add Class...
Add Method...
Add Property...
Delete
Edit Code
Data Binding...
C:\...\SplActivexCtl.h, C:\...\SplActivexCtl.cpp, C:\sample\SplActivex\SplActivex.odl
External names:
Myvar
Select a class that supports Automation and click Add Property or Add Method to add functionality to your interface.
Both Add Method and Add Property allow you to add stock and custom interfaces.
Implementation:
short m_myvar;
void OnMyvarChanged();
Default property
OK
Cancel

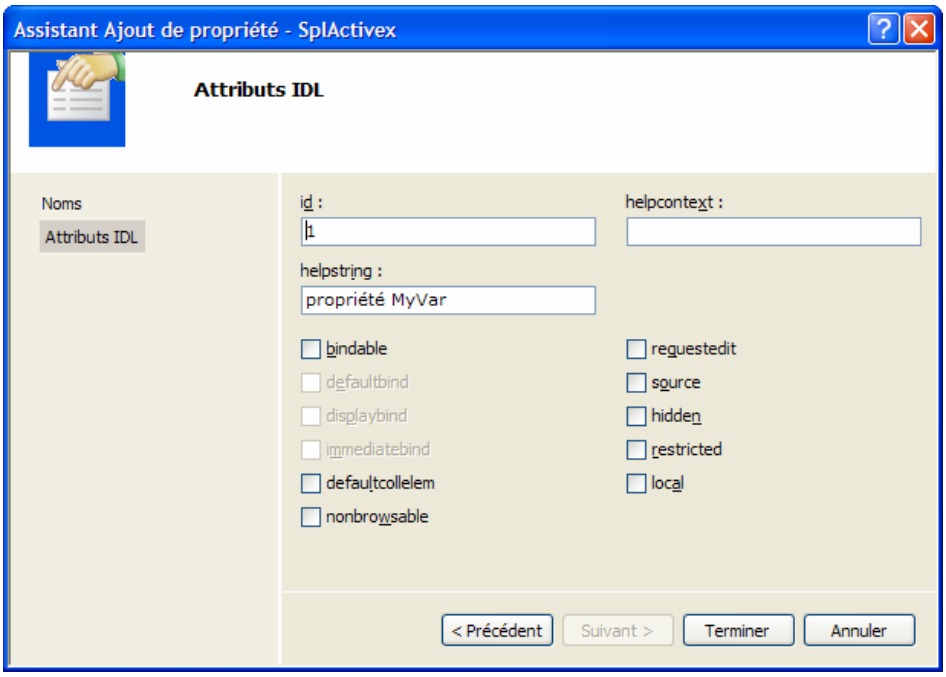
III-C. Avec Visual 2005 :

Tout se passe dans le sélecteur de classes :





Visual 2005 fait clairement la différence entre les variables du stock et les variables membres.



Par rapport à Visual 6.0 cet onglet est une nouveauté.
On pourra consulter les différentes options possibles en utilisant l'aide disponible avec le pointeur de la souris sur chacune des zones…
Enfin cliquez sur terminer.

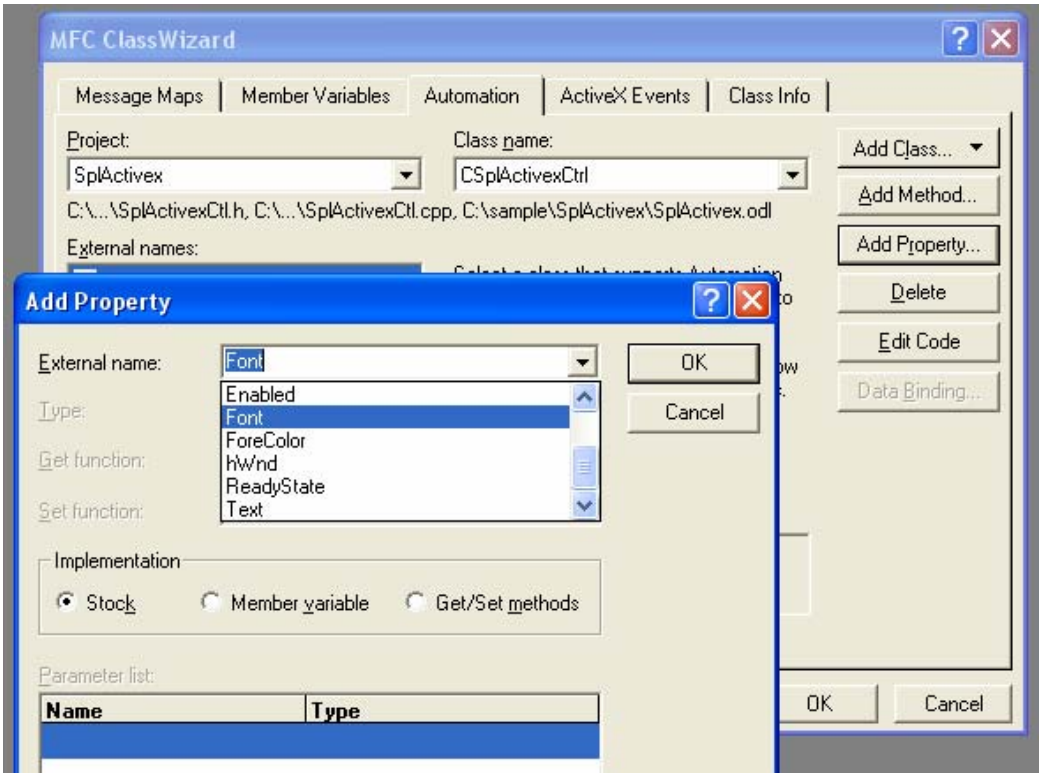
Si votre ActiveX était déjà en production il faudra gérer l'initialisation correcte des valeurs rajoutées pour les anciennes données stockées par l'ActiveX.
Mais nous verrons plus loin la sauvegarde et la gestion des versions pour l'ActiveX.

III-C-1. Ajout de propriétés particulières :

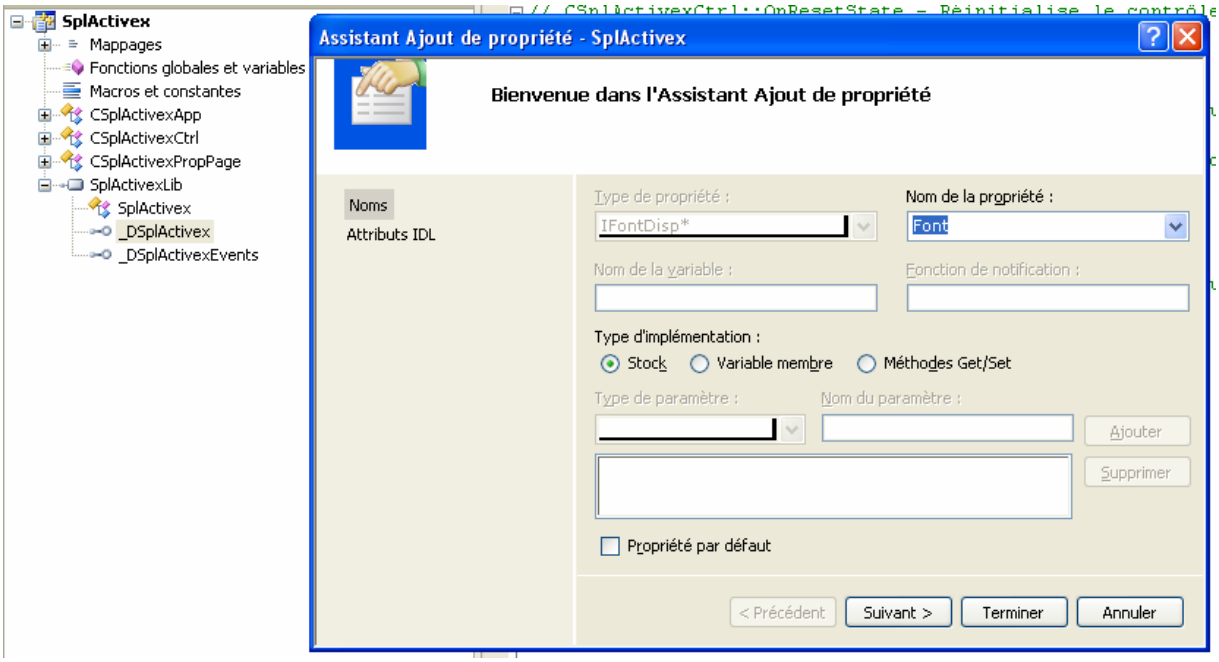
Nous allons rajouter quelques propriétés intéressantes comme la fonte, la gestion des couleurs et une image.
Le container prend en charge pour ces propriétés la gestion de leur contenu

III-C-1-A. Ajout d'une fonte

Avec l'assistant de Visual 6.0 :



Avec l'assistant de Visual 2005 :
Sélectionnez la coche stock.



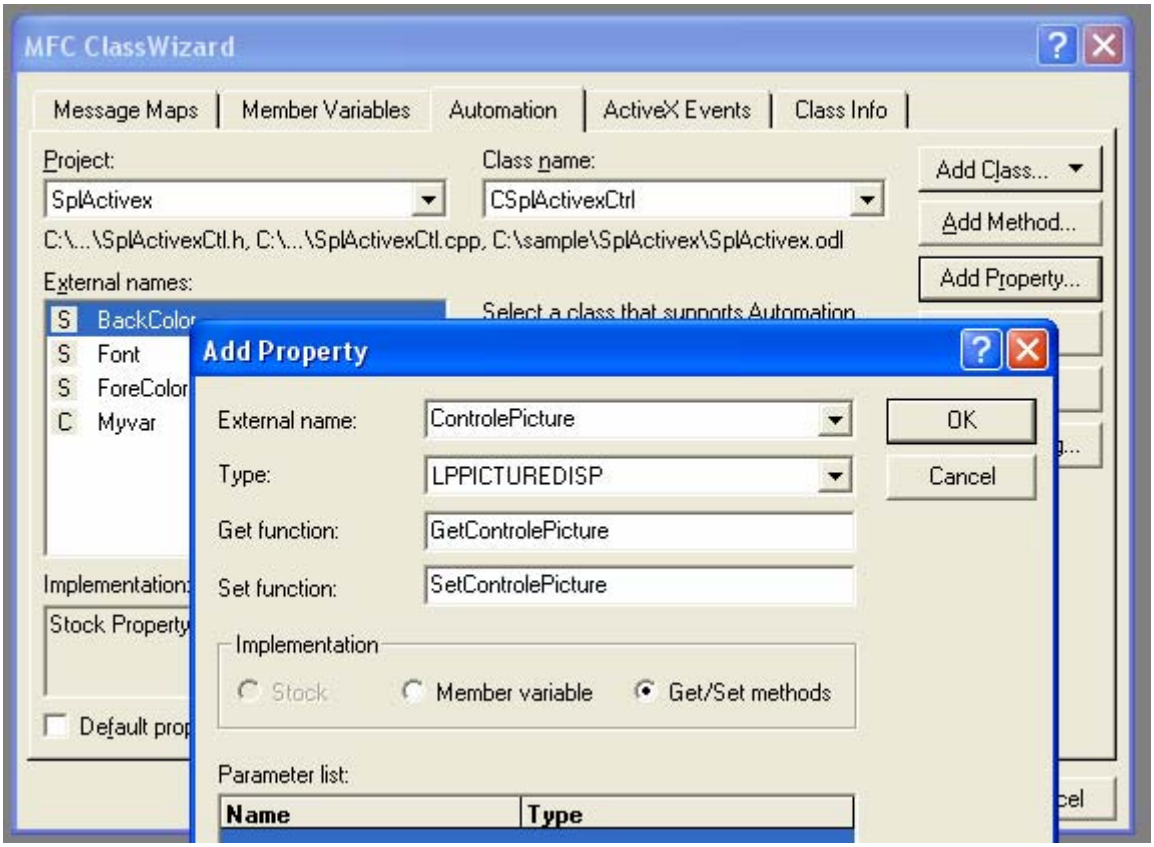
III-C-1-B. Ajout des propriétés couleurs :

Pour les couleurs le stock possède les variables **BackColor** et **ForeColor**
Procédez de la même manière que précédemment en sélectionnant les variables dans le stock.

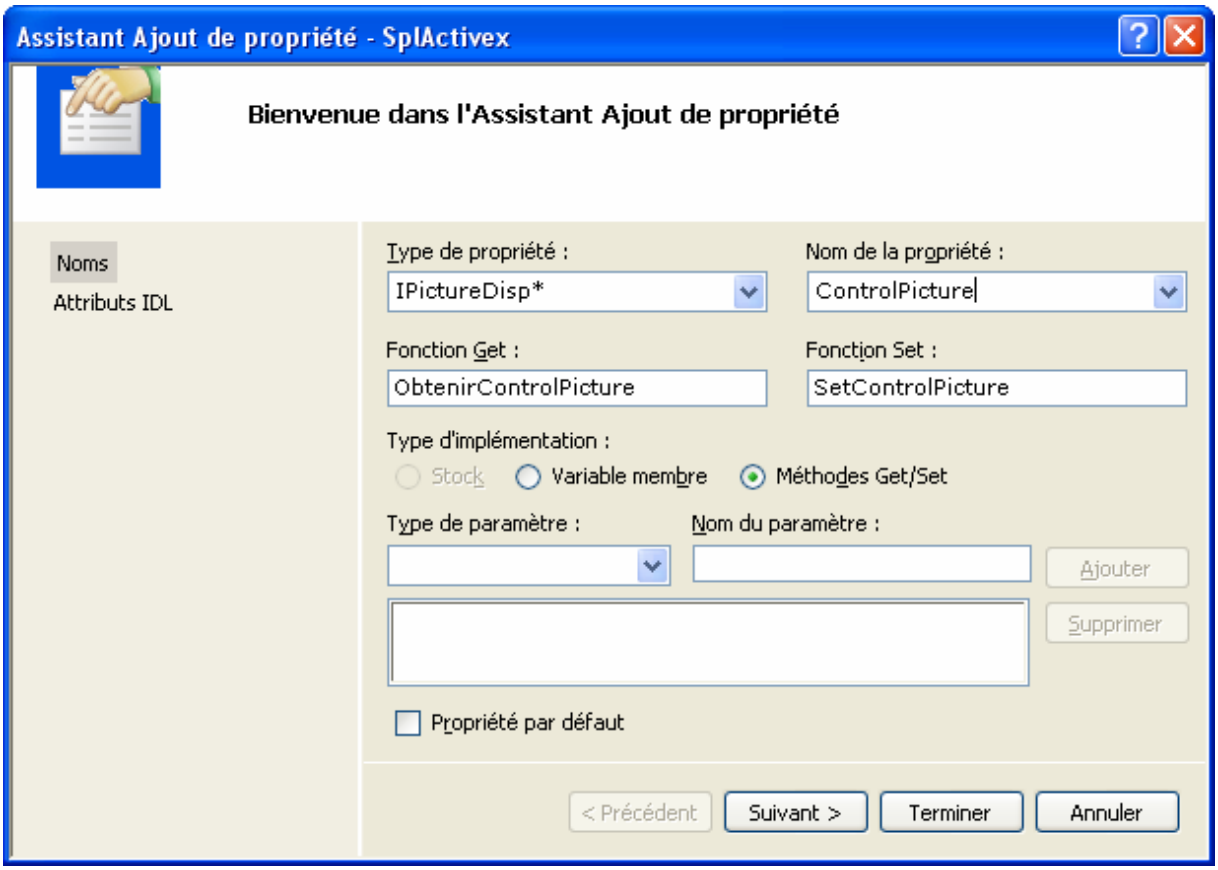
III-C-1-C. Ajout d'une image :

L'ActiveX ne possède pas de propriété image dans le stock, pour en rajouter une, procédez comme suit :

Avec l'assistant de Visual 6.0 :



Avec l'assistant de Visual 2005 :



Nous allons maintenant ajouter les pages d'interfaces pour ces propriétés qui se chargeront de gérer ces propriétés ….

Rajoutez les lignes suivantes dans la table des pages :

PROPPAGEID(CLSID_CColorPropPage)
PROPPAGEID(CLSID_CFontPropPage)
PROPPAGEID(CLSID_CPicturePropPage)

Pour obtenir :

```
// TODO : ajoutez autant de pages de propriétés que nécessaire. N'oubliez
pas d'augmenter le compteur !
BEGIN_PROPPAGEIDS(CSplActiveXCtrl, 4)
      PROPPAGEID(CSplActiveXPropPage::guid)
      PROPPAGEID(CLSID_CColorPropPage)
      PROPPAGEID(CLSID_CFontPropPage)
      PROPPAGEID(CLSID_CPicturePropPage)
END_PROPPAGEIDS(CSplActiveXCtrl)
```

Nous verrons le résultat de ces ajouts un peu plus bas …

III-C-1-D. L'utilisation de la propriété Fonte :

Pour affecter la propriété fonte dans un DC on utilisera la fonction **COLeControl::SelectStockFont** .
Exemple:

```
const CString& strCaption = InternalGetText();
pdc->SetTextColor(TranslateColor(GetForeColor()));
pdc->SetBkColor(TranslateColor(GetBackColor()));

CFont *pOldFont=SelectStockFont(pdc);
pdc->DrawText(strCaption, Rect,DT_LEFT);
pdc->SelectObject(pOldFont);
```

III-C-1-E. L'utilisation des propriétés Couleurs :

Les propriétés couleurs nécessitent d'utiliser des fonctions de conversions.
En effet les fonctions **ForeColor** et **BackColor** retournent un **OLE_COLOR** et les fonctions GDI utilisent un **COLORREF** .
On utilisera dans ce cas une fonction de conversion **COleControl::TranslateColor**
Exemple:

```
CBrush bkBrush(TranslateColor(GetBackColor()));
COLORREF clrFore = TranslateColor(GetForeColor());
pdc->FillRect( rcBounds, &bkbrush );
pdc->SetTextColor( clrFore );
pdc->DrawText( InternalGetText(), -1, rcBounds, DT_SINGLELINE | DT_CENTER |
DT_VCENTER );
```

III-C-1-F. Traitements sur la gestion de l'image:

Dans la classe de l'ActiveX déclarez une variable de la classe **CPictureHolder** :

```
class CSplActiveXCtrl : public COleControl
{
	DECLARE_DYNCREATE(CSplActiveXCtrl)

// Constructeur
public:
	CSplActiveXCtrl();

private:
	CPictureHolder	m_Pic;
```

Ensuite nous allons remplir les fonctions générées pour notre variable `ControlePicture`

```
IPictureDisp* CSplActiveXCtrl::ObtenirControlePicture(void)
{
	AFX_MANAGE_STATE(AfxGetStaticModuleState());

	// TODO : ajoutez ici le code de votre gestionnaire de dispatch
    return m_Pic.GetPictureDispatch();
}

void CSplActiveXCtrl::SetControlePicture(IPictureDisp* pVal)
{
	AFX_MANAGE_STATE(AfxGetStaticModuleState());

	// TODO : ajoutez ici le code de votre gestionnaire de propriété
	m_Pic.SetPictureDispatch(pVal);
	SetModifiedFlag();
	InvalidateControl();
}
```

Nous verrons plus tard comment sauvegarder notre variable **m_Pic**.

Affichage de l'image dans la fonction **OnDraw** :

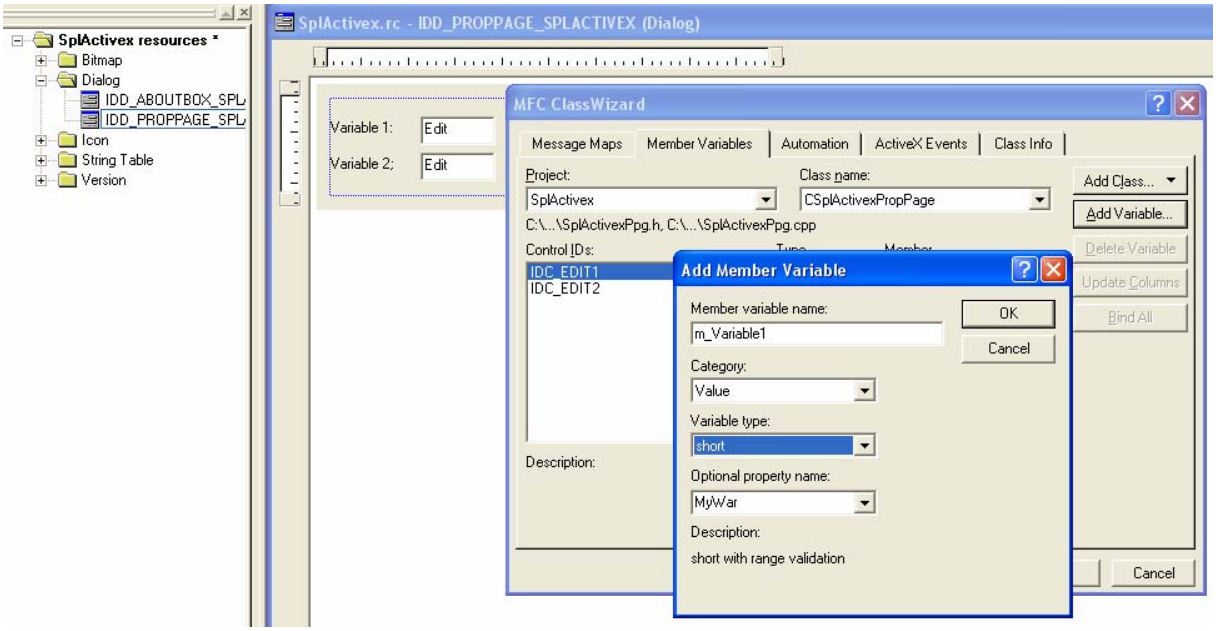
```
// affichage de l'image en cours de selection
CString strValue="";
if(m_Pic.GetType()!=PICTYPE_UNINITIALIZED &&
  m_Pic.GetDisplayString(strValue)	&&
  strValue!="(Picture - None)" && strValue!="(Image - Aucun)")
		m_Pic.Render(pdc,  rcBounds, rcBounds);
```

III-C-1-G. Définition de l'interface pour les variables :

Nous venons de voir les interfaces disponibles pour les propriétés particulières comme les couleurs, les fontes, les images ; il nous reste à voir l'interfaçage des autres variables (custom).

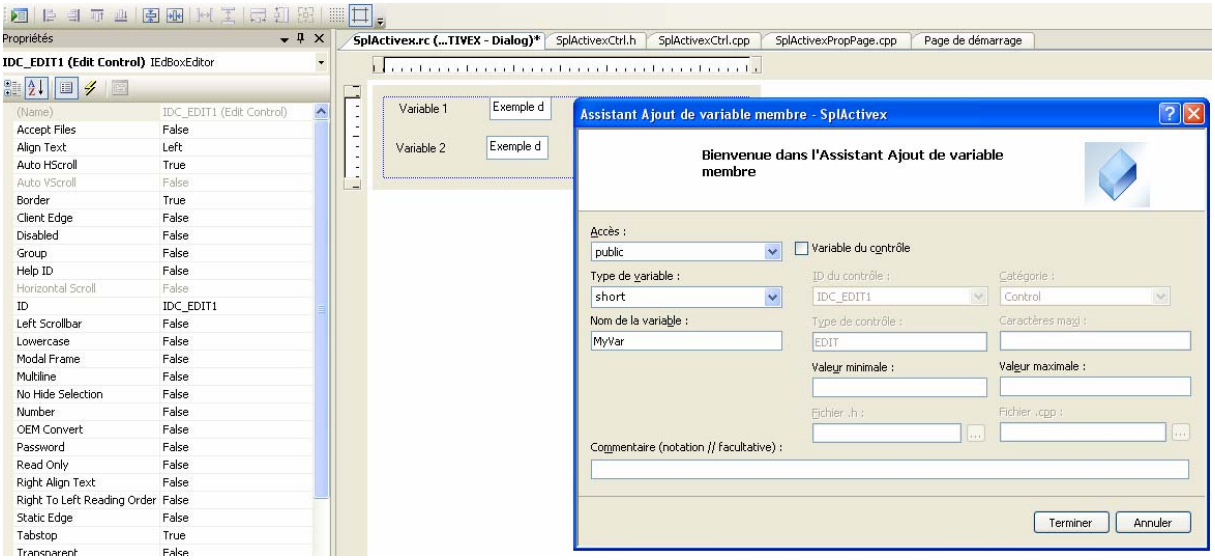
Nous allons interfacer la variable que nous avons créée au début de ce chapitre.
Pour cela placez-vous dans les ressources sur la boîte de dialogue générée par l'assistant.
L'ajout d'une variable se passe de la même manière que pour un projet MFC classique, on place le contrôle sur la fenêtre et on l'associe à une variable correspondant à son type défini

Avec l'assistant de Visual 6.0 :



Le lien avec la propriété est établi en inscrivant manuellement le nom de la variable dans le sélecteur **« optional property name ».**
Il vous faudra donc noter le nom de la variable avant d'utiliser ClassWizard, car le sélecteur ne propose que les variables de stock.

Avec l'assistant de Visual 2005 :



L'association de la variable se passe comme pour une variable d'un projet MFC classique, Mais là où Visual 6.0 permettait d'associer directement la variable à la propriété, Visual 2005 fait l'impasse sur ce sujet, ce que je trouve regrettable (bug d'oubli ?).
En tout cas cela va nous obliger à effectuer ce lien manuellement dans la fonction **DoDataExchange** de notre page de propriété.

```
// CSplActiveXPropPage::CSplActiveXPropPage - Constructeur
CSplActiveXPropPage::CSplActiveXPropPage() :
	COlePropertyPage(IDD, IDS_SPLACTIVEX_PPG_CAPTION)
		, MyVar(0)
		, MyVarTwo(0)
		, m_strCaption(_T(""))
	{
}
// CSplActiveXPropPage::DoDataExchange - Transfère les données entre la
page et les propriétés

void CSplActiveXPropPage::DoDataExchange(CDataExchange* pDX)
{
	DDP_Text(pDX, IDC_EDIT1, m_MyVar, _T("MyVar") );
	DDP_Text(pDX, IDC_EDIT2, m_MyVarTwo, _T("MyVarTwo") );
	DDP_Text(pDX, IDC_EDIT3, m_strCaption,"Text" );
	DDP_PostProcessing(pDX);
}
```

Le lien s'effectue donc au moyen de la macro **DDP_Text** qui supporte plusieurs signatures suivant le type de variable utilisée (float,double,short,int ,CString etc).
Le dernier argument de la fonction étant le nom de la propriété de l'ActiveX.
Il existe d'autres macros comme **DDP_Radio** et **DDP_Check**.
Voir ce lien MSDN pour la liste des macros : http://msdn2.microsoft.com/ko-kr/library/y2y0fsk8.aspx

En résumé :
Pour chaque propriété custom ou du stock vous pouvez définir un accès pour la renseigner à travers une page de propriétés.
Ceci s'applique à toutes les propriétés hormis les couleurs et les fontes puisqu'elles disposent de leur propre interface prédéfinie.
La macro **DDP**_ permet de faire l'association directe entre la variable de votre page de propriétés et celle définie dans l'ActiveX.
Les lignes de description **DDP**_ doivent être placées impérativement avant la ligne **DDP_PostProcessing**.

Voilà vous connaissez maintenant les techniques pour définir des propriétés et leur interface de saisie.

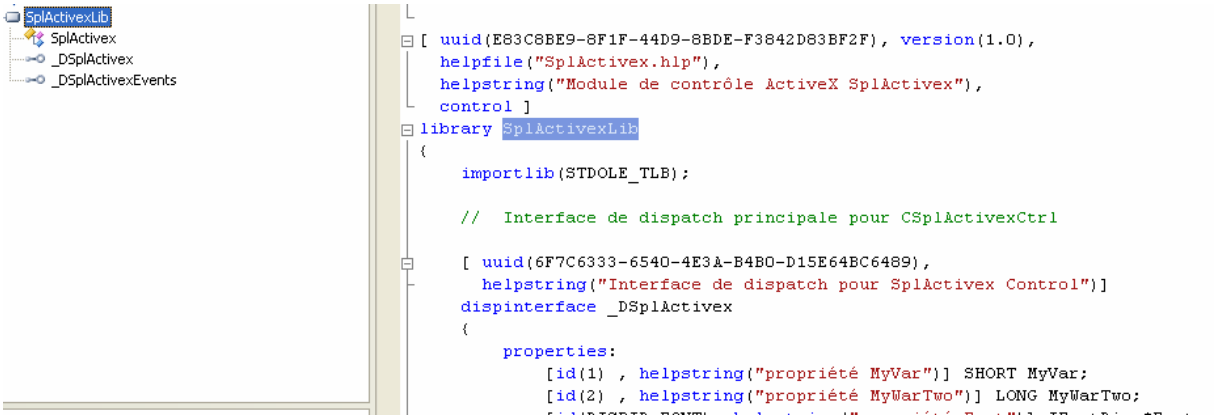
Note :
Pour une raison que j'ignore lorsque l'on génère un projet ActiveX avec Visual 2005 la modification des variables personnalisées (custom) dans la boîte de propriétés ne fonctionne pas.
Par contre un projet construit à partir de Visual 6.0 et converti en Visual 2005 ne pose pas de problème.
Après avoir passé plusieurs heures sur le sujet pour les besoins de ce tutoriel je n'ai pas de réponse.

III-C-1-H. Suppression d'une méthode ou variable de l'ActiveX

Une dernière remarque pour vous dire que j'ai constaté un dernier manque à la gestion des propriétés et méthodes de l'ActiveX avec Visual 2005 : la suppression des propriétés ou méthodes.

Là ou l'assistant (ClassWizard) de Visual 6.0 permet au moyen d'un bouton de s'acquitter de cette tache, il semblerait que Visual 2005 ne nous laisse pour option que de faire les choses manuellement.
Sauf erreur de ma part je n'ai pas trouvé dans l'interface de mécanisme similaire à Visual 6.0

Comment faire ?:
Il faudra éditer manuellement le fichier **.idl** (interface définition file) de l'ActiveX (double clic sur l'interface)
Et supprimer manuellement la ligne dans l'interface.

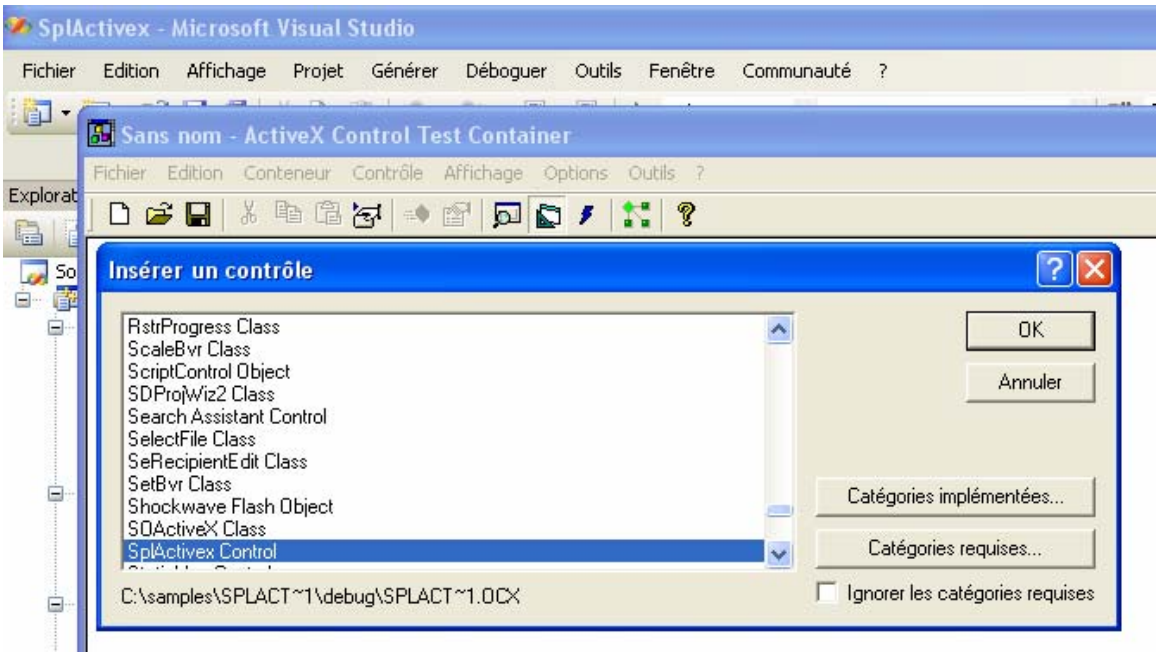


Il faudra ensuite fermer le projet pour supprimer le fichier **.ncb**, ré-ouvrir le projet.
Et dans la table des propriétés il faudra supprimer la ligne correspondante.

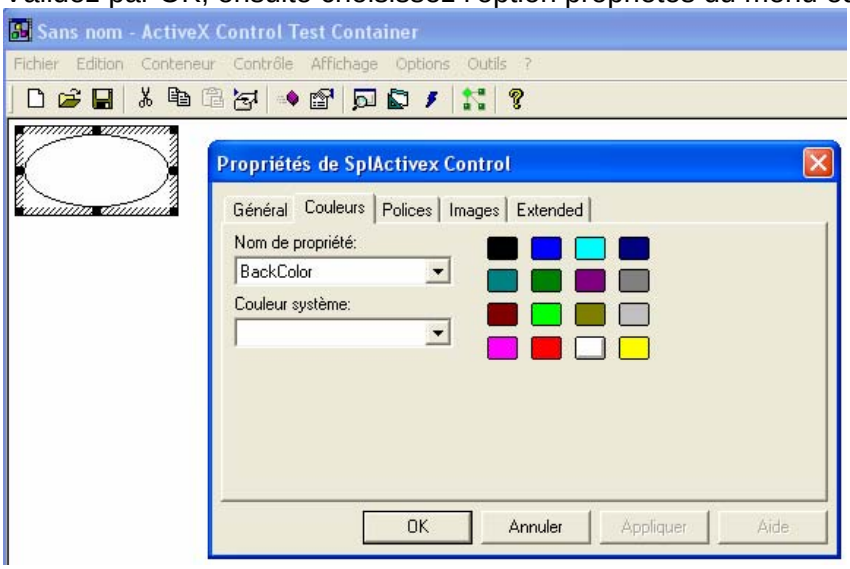
```
BEGIN_DISPATCH_MAP(CSplActiveXCtrl, COleControl)
	//{{AFX_DISPATCH_MAP(CSplActiveXCtrl)
	DISP_PROPERTY_NOTIFY(CSplActiveXCtrl, "MyVar", m_myVar, OnMyVarChanged, VT_I2)
	DISP_PROPERTY_NOTIFY(CSplActiveXCtrl, "MyVarTwo", m_myVarTwo, OnMyVarTwoChanged,
VT_I2)
	DISP_PROPERTY_EX(CSplActiveXCtrl, "ControlePicture", GetControlePicture, SetControlePicture,
VT_PICTURE)
	DISP_STOCKPROP_FONT()
	DISP_STOCKPROP_FORECOLOR()
	DISP_STOCKPROP_BACKCOLOR()
	//}}AFX_DISPATCH_MAP
	DISP_FUNCTION_ID(CSplActiveXCtrl, "AboutBox", DISPID_ABOUTBOX, AboutBox, VT_EMPTY,
VTS_NONE)
END_DISPATCH_MAP()
```

III-D.Test de l'ActiveX :

On pourra tester l'ActiveX avec le programme **ActiveX Control Test Container** disponible dans le menu outils.
A partir du menu édition utilisez l'option ajouter un nouveau contrôle.

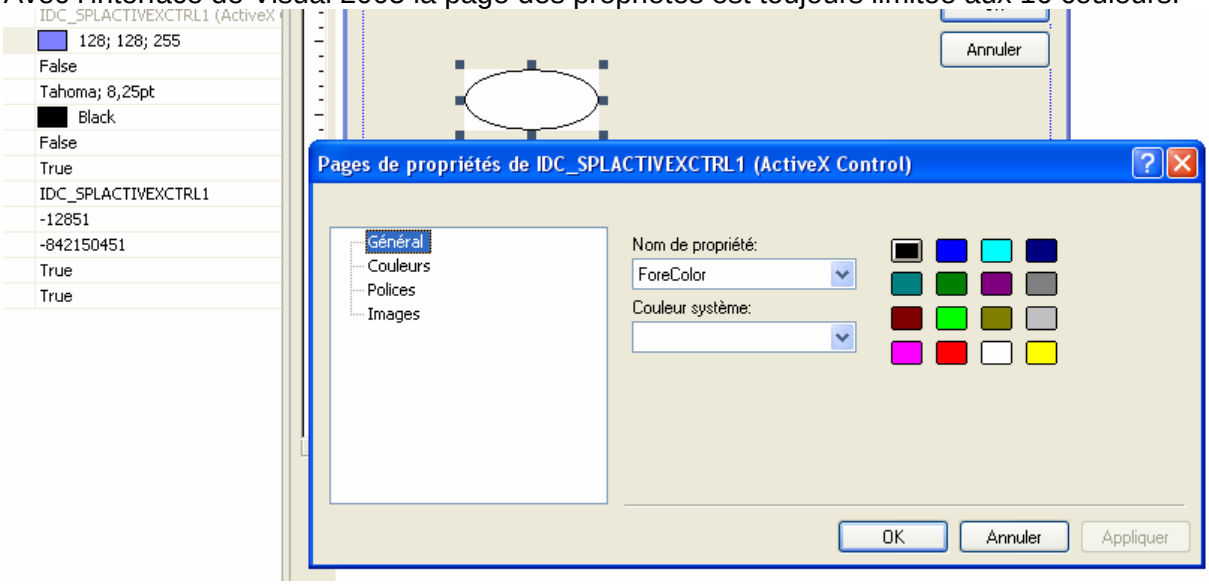


Validez par OK, ensuite choisissez l'option propriétés du menu édition.

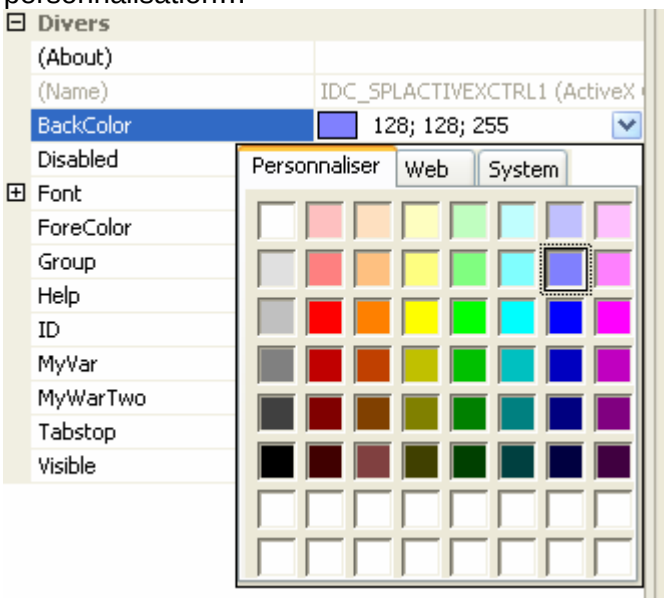


Une remarque liée à la gestion des couleurs, la gestion du conteneur sous Visual 6.0 ne nous donne pas le plein accès à la définition des couleurs !
Nous avons droit à une palette réduite, alors que dans un environnement Visual Basic on dispose pour le même ActiveX de la gestion complète des couleurs un comble !

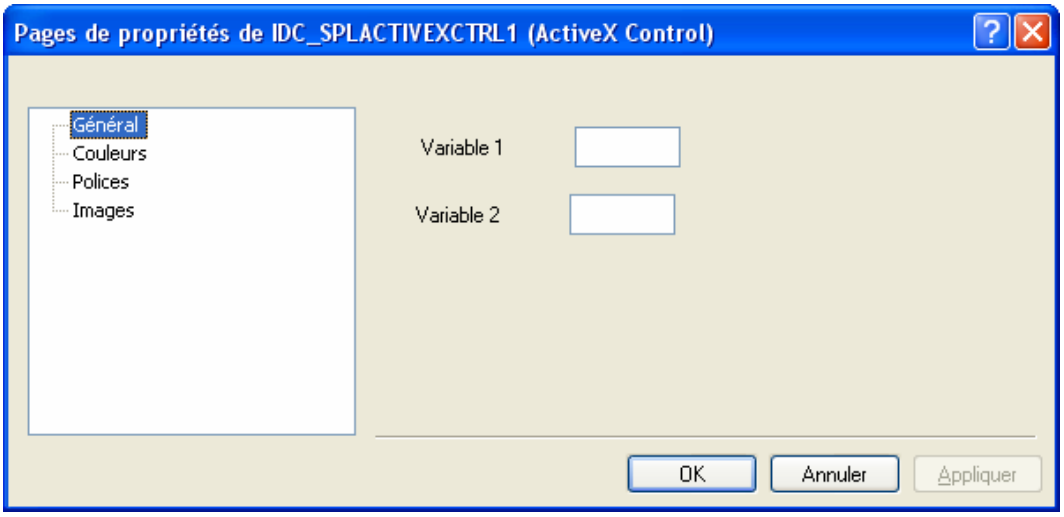
Avec l'interface de Visual 2005 la page des propriétés est toujours limitée aux 16 couleurs.

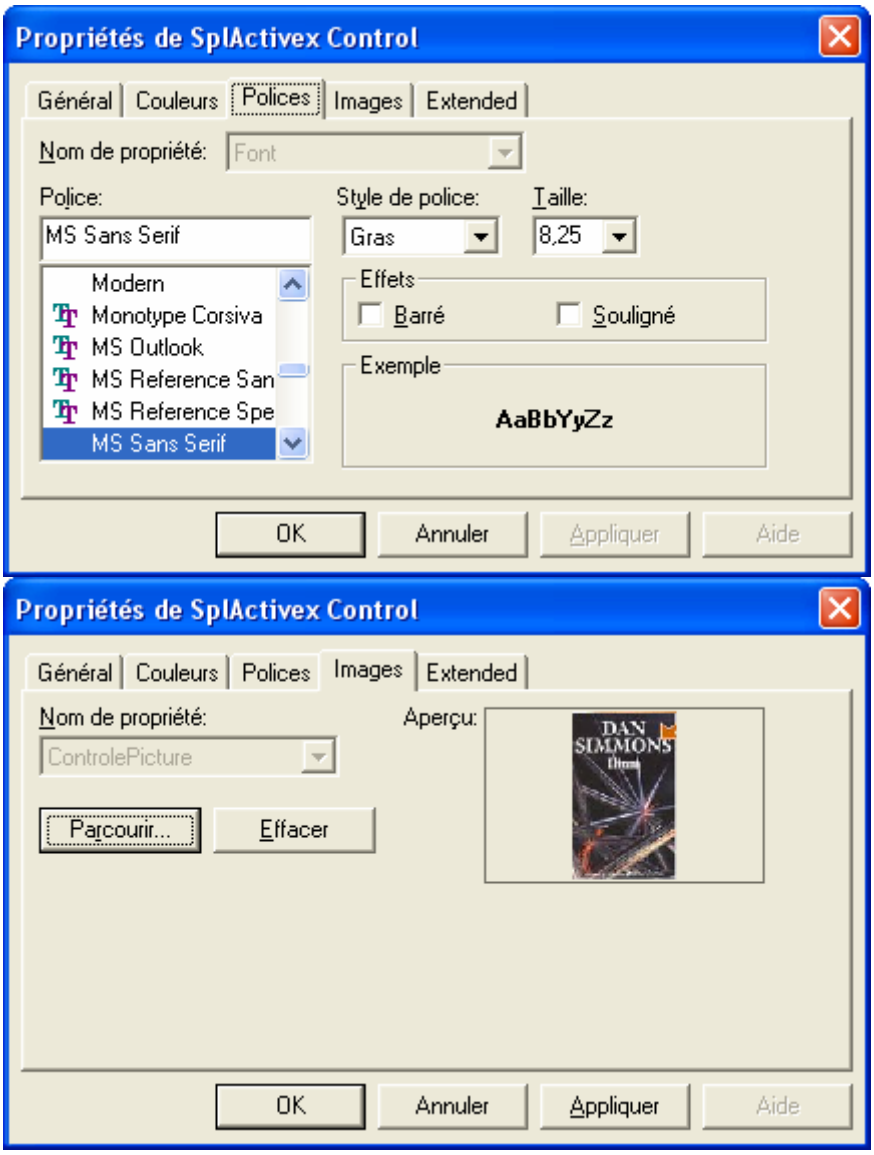


Par contre en passant par le tableau des propriétés on dispose de la palette complète. Pour accéder à la personnalisation de la couleur il faudra faire clic droit sur les zones de personnalisation…



Interface avec Visual 2005 des pages de propriétés dans un projet :





Voilà avec peu d'effort nous avons implémenté les interfaces d'accès à ces variables.

IV. Ajout d'une méthode dans l'ActiveX

IV-A . Qu'est ce qu'une méthode dans un ActiveX ?

Ce sont des fonctions que vous déclarez au niveau de l'ActiveX et qui seront disponibles dans l'interface utilisateur générée (wrapper de classe) .
Comme pour les propriétés on utilisera l'assistant pour les définir.

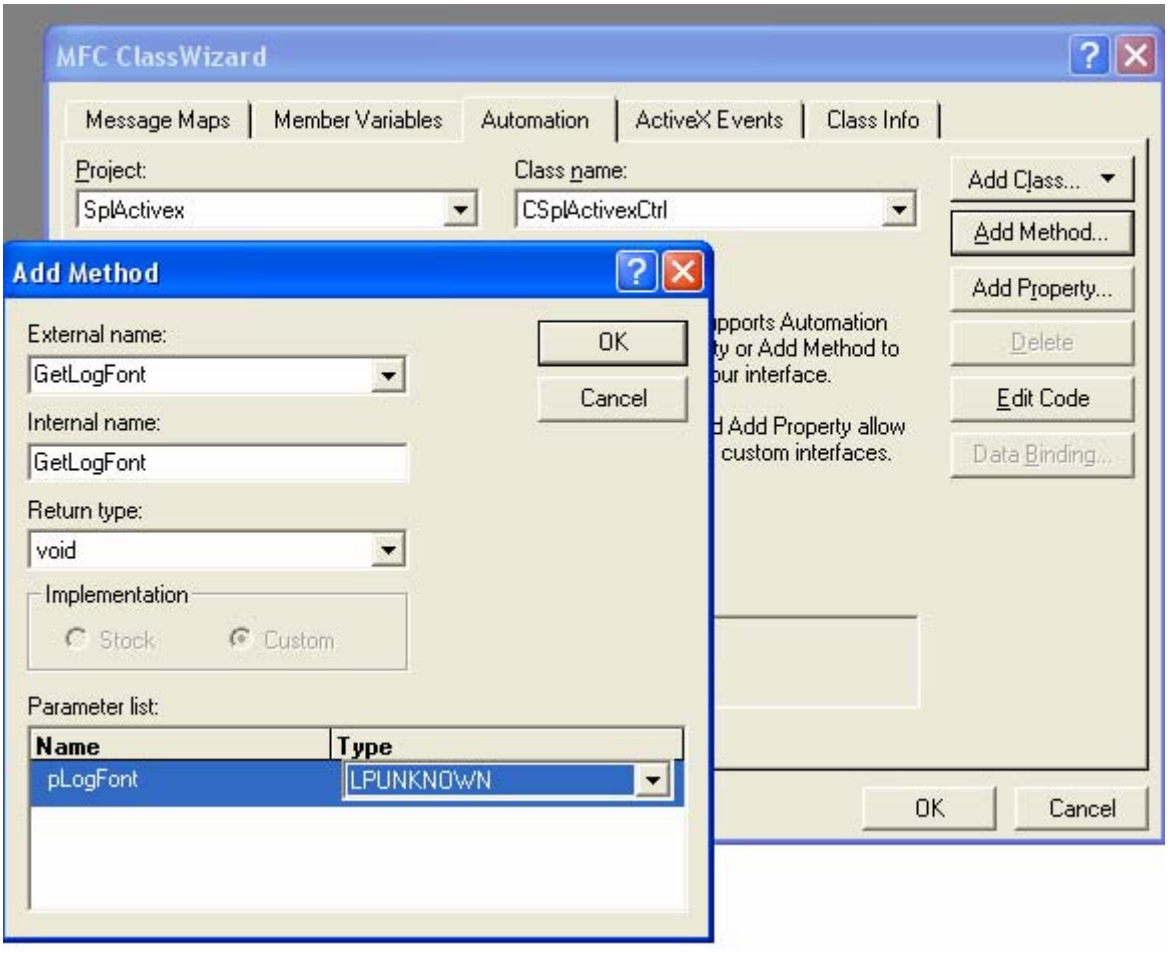
IV-A-1. Une méthode pour quoi faire ?

On a vu que pour une propriété une interface d'accès (fonctions Get et Set) était générée dans la classe Wrapper,
Une méthode permettra d'appeler un traitement particulier dans l'ActiveX ou de transférer des données dans les deux sens : du programme vers l'ActiveX ou de l'ActiveX vers le programme.

Pour rajouter une méthode sur un ActiveX on utilisera l'assistant qui varie selon les versions de Visual.

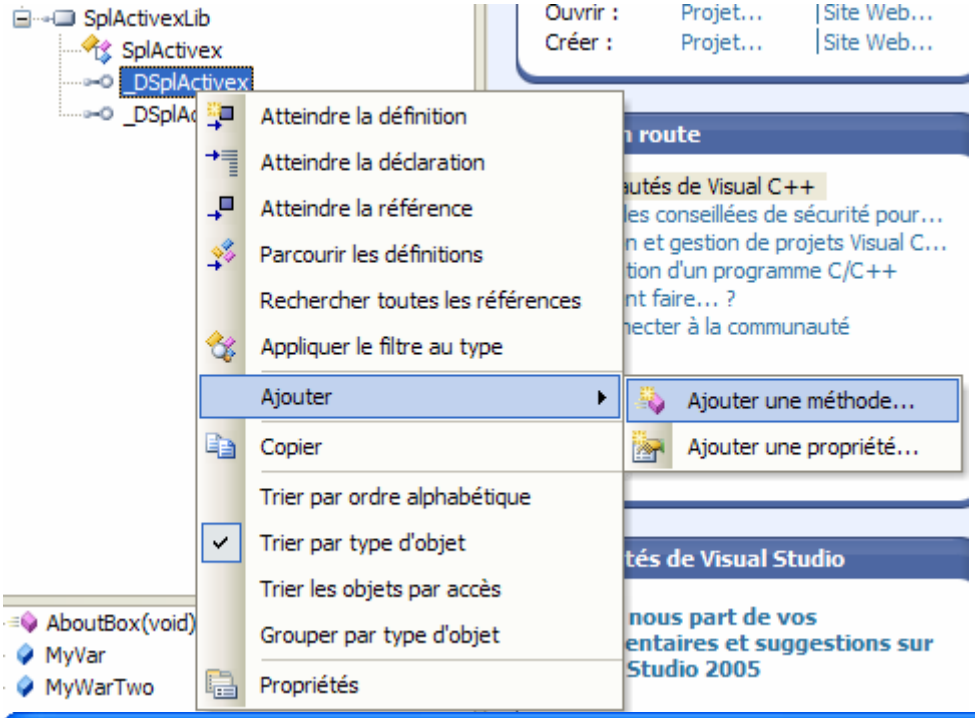
Pour illustrer mon propos je vais créer une méthode qui me permettra de récupérer la structure **LOGFONT** correspondant à la fonte en cours dans l'ActiveX.

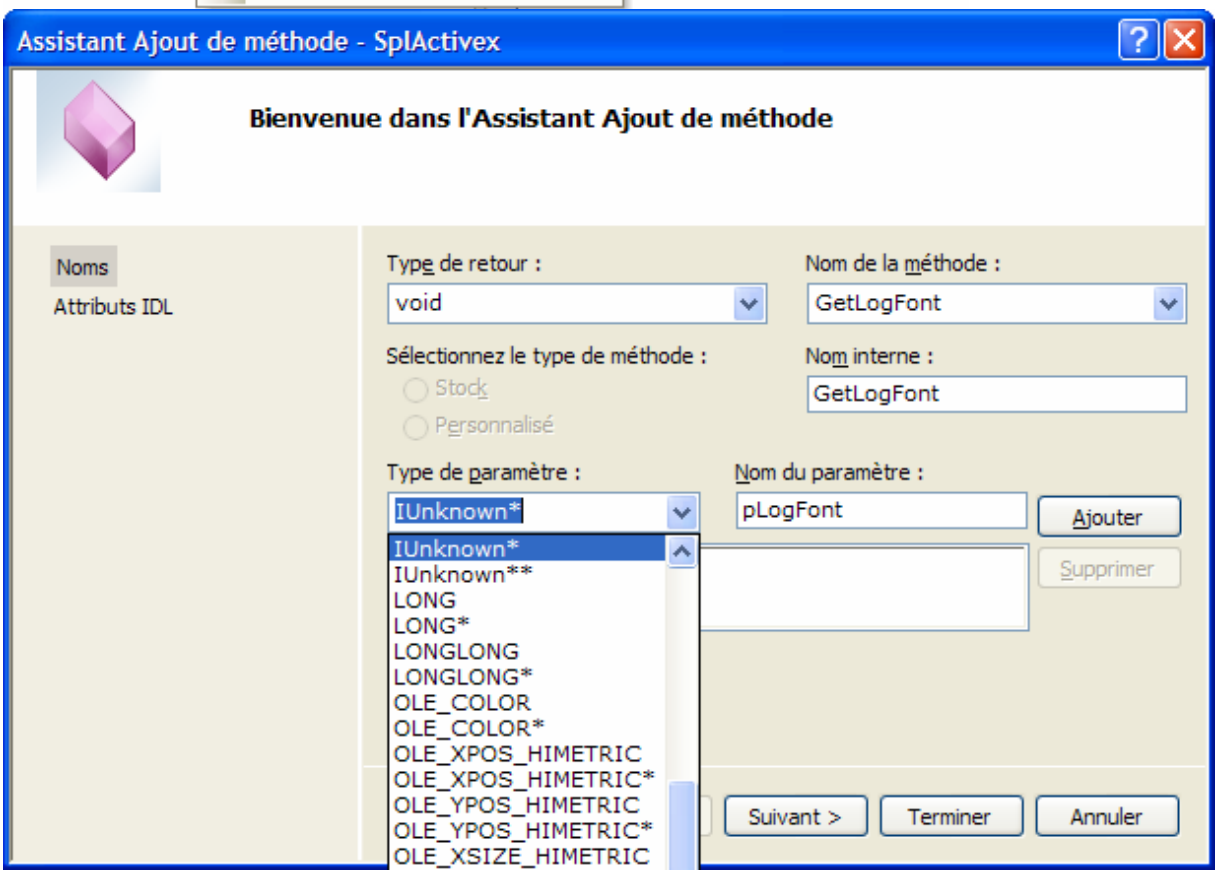
IV-B. Avec Visual 6.0 :



IV-C. Avec Visual 2005 :

Comme précédemment tout se passe dans le sélecteur de classes :





Premier problème je ne dispose pas de ce type dans le sélecteur de paramètres, je vais donc me rabattre sur un pointeur inconnu pour recevoir un pointeur sur une structure **LOGFONT**.

Quand l'argument est paramétré il suffit de cliquer sur le bouton Ajouter.
Cliquez ensuite sur le bouton Terminer.

Voici le code généré :

```
void CSplActiveXCtrl::GetLogFont(IUnknown* pLogFont)
{
	AFX_MANAGE_STATE(AfxGetStaticModuleState());

	// TODO : ajoutez ici le code de votre gestionnaire de dispatch
}
```

Et le traitement effectué :

```
void CSplActiveXCtrl::GetLogFont(IUnknown* pLogFont)
{
	AFX_MANAGE_STATE(AfxGetStaticModuleState());
	// TODO : ajoutez ici le code de votre gestionnaire de dispatch
	LOGFONT lg;
	CFontHolder *pHolder=&InternalGetFont();
	if(pHolder)
	{
		CFont *pFont = CFont::FromHandle(pHolder->GetFontHandle())
		if(pFont)
		{
			pFont->GetLogFont(&lg);
			memmove(pLogFont,&lg,sizeof(lg));
		}
	}
}
```

V. Ajout d'un événement dans l'ActiveX

V-A . Qu'est ce qu'un événement dans un ActiveX ?

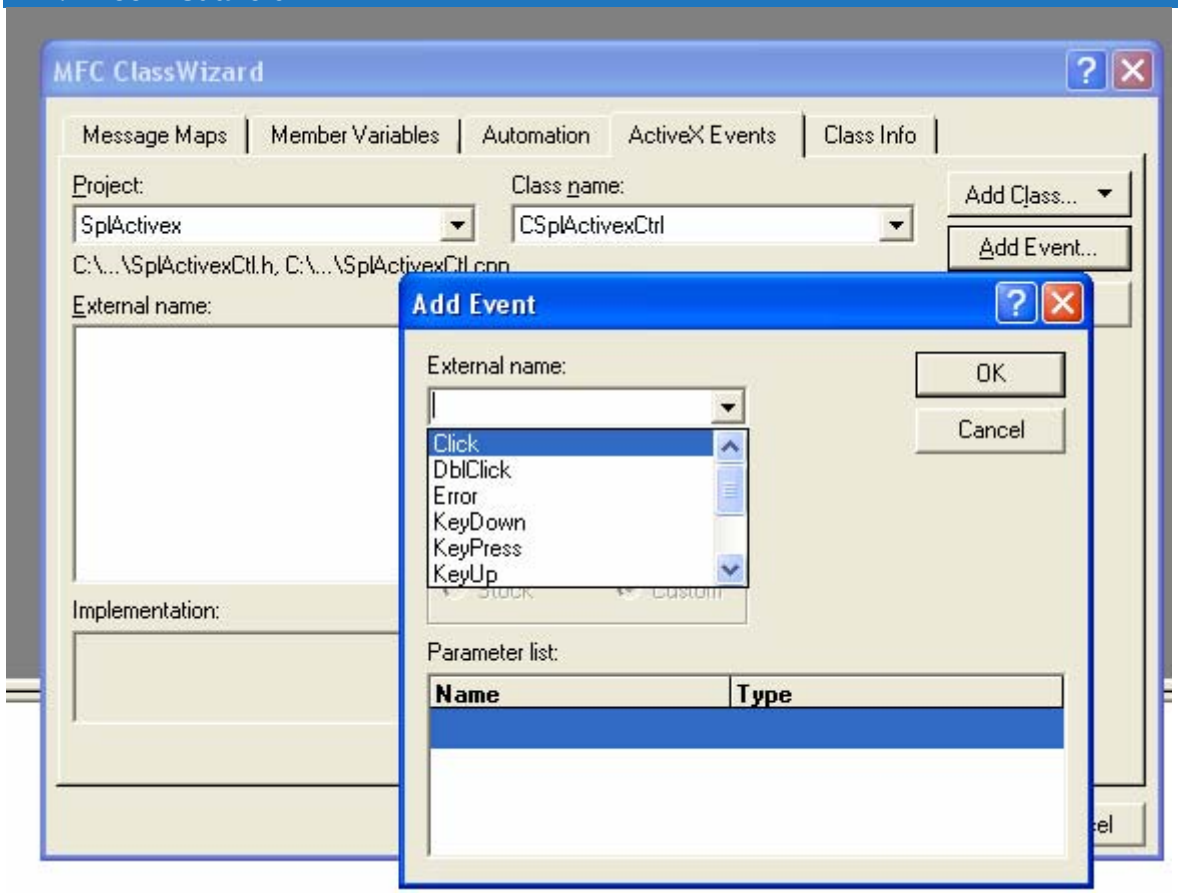
Ce sont des événements souris ou clavier que votre ActiveX prendra en compte.
Ces événements seront donc disponibles et interceptables par l'utilisateur de l'ActiveX.
Les événements disponibles dans le stock permettent de traiter la majorité des cas :
Clic, double clic, touche pressée, relâchée etc...
Si malgré cela les événements du stock ne suffisaient pas, vous pouvez définir votre propre événement personnalisé.
Les événements du stock sont des fonctions définies dans la classe **COleControl** exemple : **FireClick()**.
Il sera donc tout à fait possible d'appeler une de ces fonctions directement dans l'ActiveX pour simuler un événement clavier ou souris…

Comme pour les propriétés on utilisera l'assistant pour les définir.

V-A-1. Un événement pour quoi faire ?

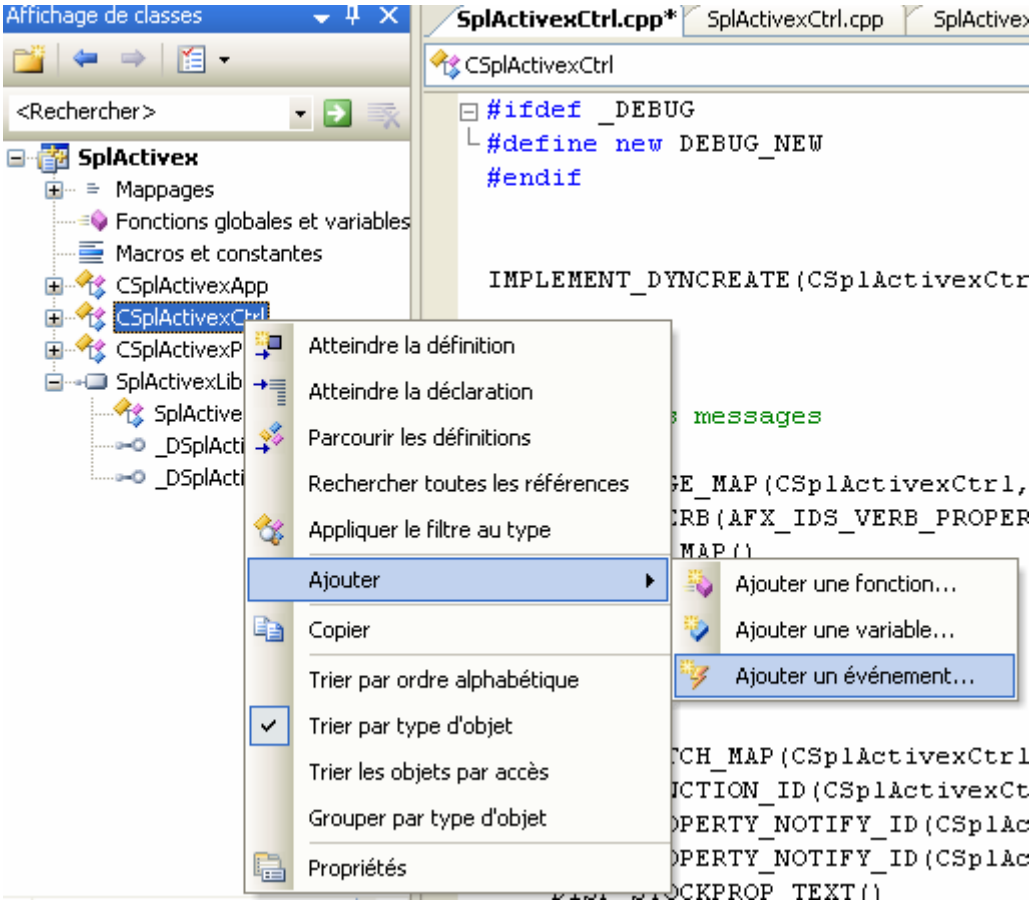
Si votre ActiveX est destiné à implémenter un nouveau contrôle il vous faudra donner la possibilité à l'utilisateur de traiter les messages souris ou clavier de ce contrôle.
Les événements personnalisés permettront de traiter les messages de type **OnChange**
Comme sur un **CEdit** ou encore un **OnSelChange** comme sur une **CListBox** etc…

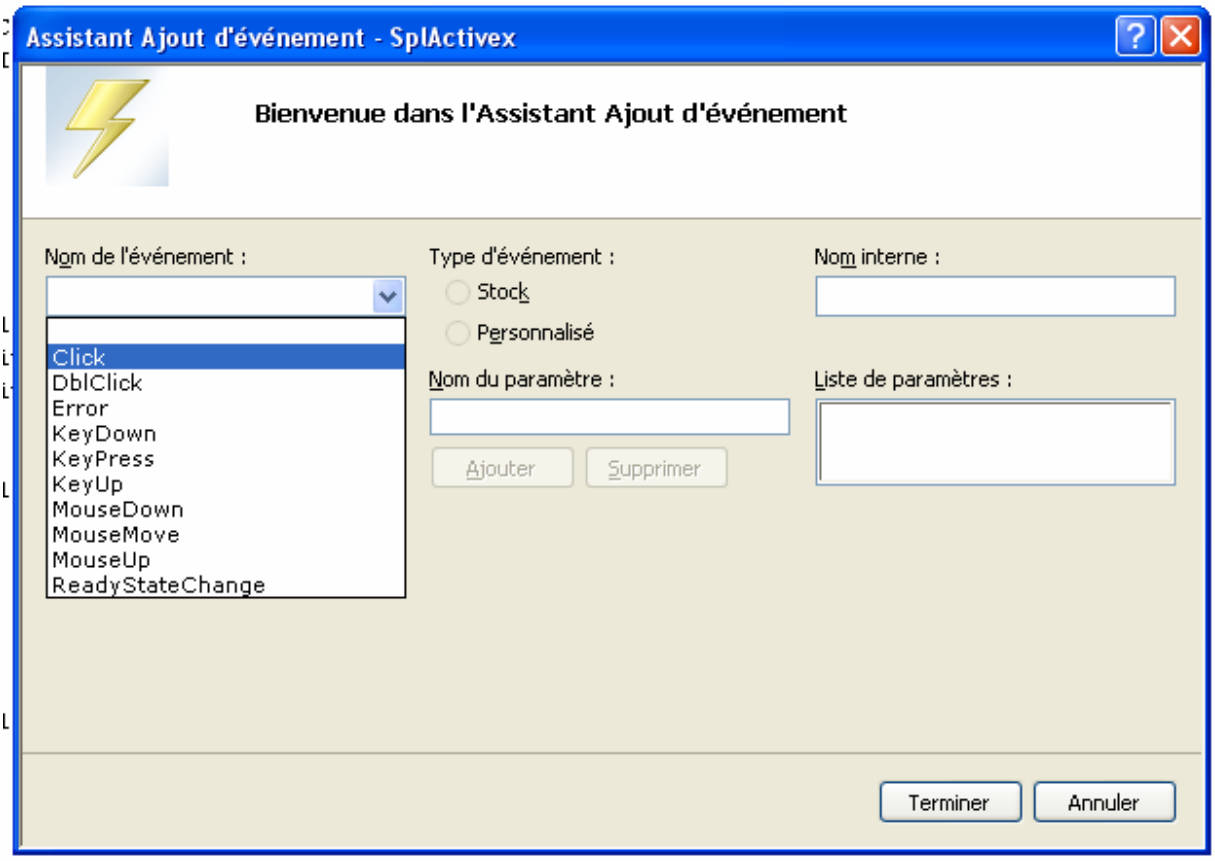
V-B. Avec Visual 6.0 :



V-C. Avec Visual 2005 :

On utilisera l'assistant ajout d'un événement à partir de la **classe du contrôle** :





L'assistant générant le code suivant dans la table des événements :

```
// Table d'événements

BEGIN_EVENT_MAP(CSplActiveXCtrl, COleControl)
      EVENT_STOCK_CLICK()
END_EVENT_MAP()
```

VI. Persistance des données de l'ActiveX

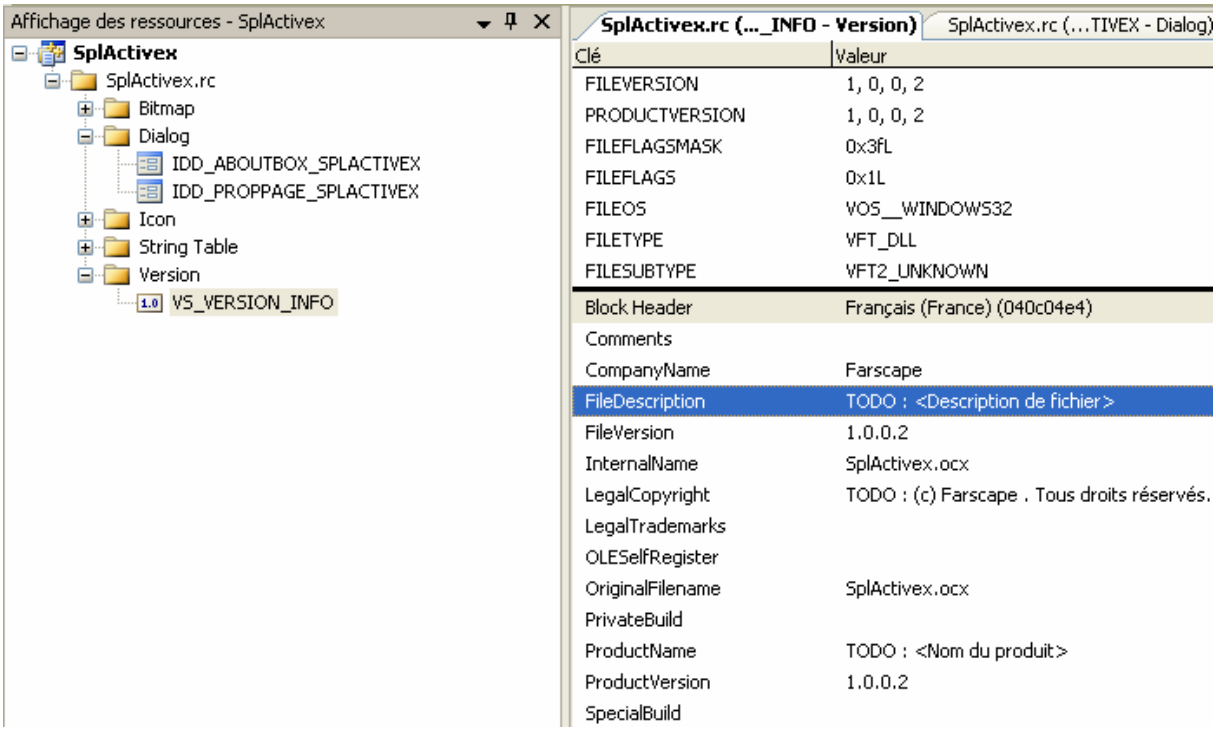
Nous avons vu comment déclarer des propriétés à notre ActiveX, il nous reste à voir comment traiter la persistance de ces données.
Le traitement de lecture et écriture des données est à faire dans la fonction **DoPropExchange(CPropExchange* pPX) .**
Exemple de traitement :

```
void CSplActiveXCtrl::DoPropExchange(CPropExchange* pPX)
{
	ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
	COleControl::DoPropExchange(pPX);

	// TODO : appelez les fonctions PX_ pour chaque propriété
personnalisée persistante.
	CPictureHolder Pic;
	Pic.CreateEmpty();

	CString *pstrCaption =const_cast<CString *>(&InternalGetText());
	PX_String(pPX,_T("Text"),*pstrCaption,_T("Essai"));
	// exemple pour une variable de couleur supplementaire
	//PX_Color(pPX,_T("BorderColor"),m_borderColor,12632256);

	PX_Picture(pPX,_T("ControlePicture"),m_Pic,Pic);
    PX_Short(pPX,_T("MyVar"),m_MyVar,0);
   //placez ici la lecture des variables liées à l'evolution des versions
de l'ActiveX.
// le controle des numeros de version est à faire pour s'assurer de la
comptabilité des differentes versions...
	if(pPX->GetVersion()>= (DWORD)MAKELONG(1, 2))
	{
		PX_Long(pPX,_T("MyVarTwo"),m_MyVarTwo,0);
	}
	else
	if(pPX->IsLoading())
	{
		// initialisations par defaut des zones supplementaires.
		m_VarTwo=0;
	}
}
```

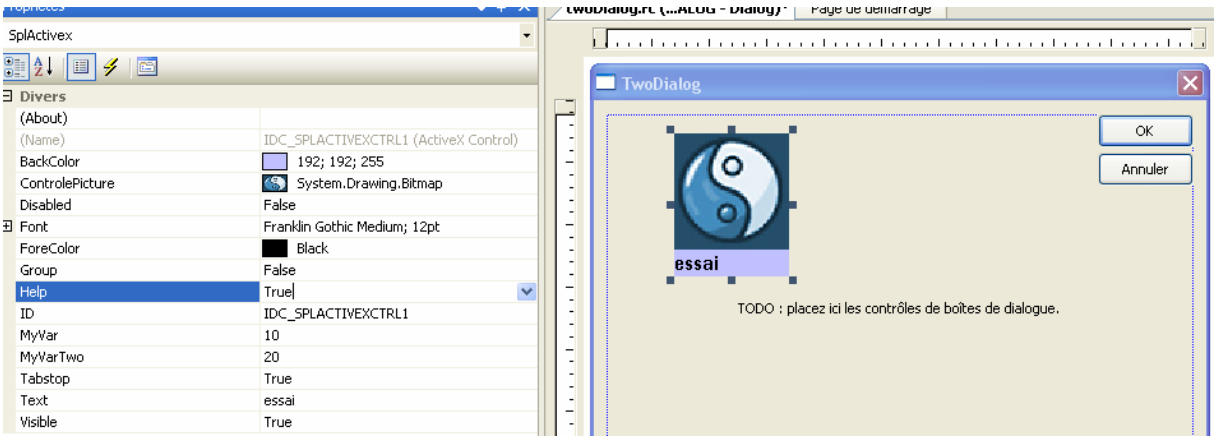
La lecture et l'écriture des données se font par l'intermédiaire des macros **PX_**.
Il en existe pour tous les types de variables possibles.
Voir MSDN pour la liste complète : http://msdn2.microsoft.com/de-de/library/h0d9tf57.aspx
Le dernier argument de cette macro permet aussi d'affecter une valeur par défaut à la propriété.

Comme vous avez dû le remarquer les variables du stock n'ont pas besoin d'être prises en charge dans cette fonction, sauf pour affecter une valeur par défaut à la variable.

La gestion des versions est assurée par la fonction **GetVersion()** qui permet de retrouver la version en cours des données de l'ActiveX.
Le numéro de version doit être défini dans les ressources de l'ActiveX.



Pour finir voici le résultat final de notre ActiveX dans l'interface utilisateur :



Le code de la fonction OnDraw qui tient compte des propriétés:

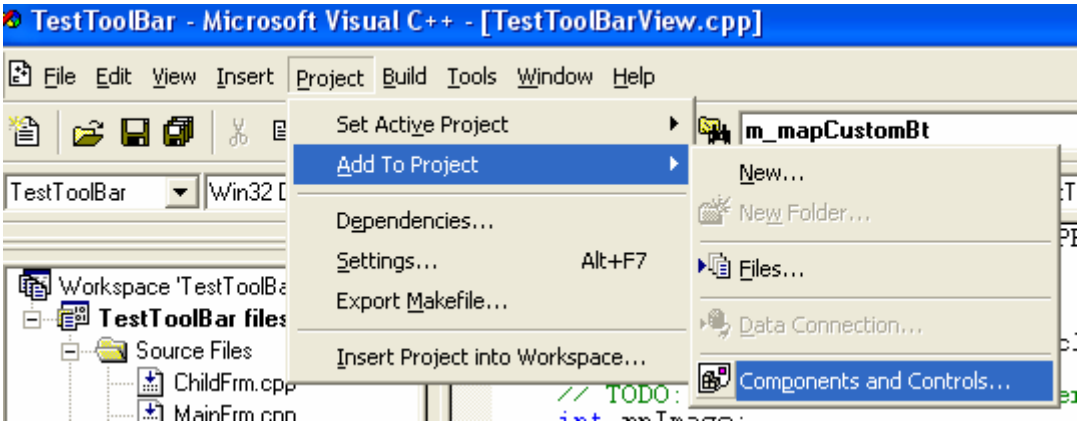
```
// CSplActiveXCtrl::OnDraw - Fonction de dessin
void CSplActiveXCtrl::OnDraw(
                  CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
      if (!pdc)   return;

      const CString& strCaption = InternalGetText();
      pdc->SetTextColor(TranslateColor(GetForeColor()));
      pdc->SetBkColor(TranslateColor(GetBackColor()));
      CBrush TextBrush(TranslateColor(GetBackColor()));
      pdc->FillRect(rcBounds,&TextBrush);
      pdc->SetBkMode(OPAQUE);

      CFont *pOldFont=SelectStockFont(pdc);
      CSize Size=pdc->GetTextExtent(strCaption);

      CRect Rect=rcBounds;
      Rect.bottom-=Size.cy;

      // affichage de l'image en cours de selection
      CString strValue=_T("");
      if(m_Pic.GetType()!=PICTYPE_UNINITIALIZED &&
         m_Pic.GetDisplayString(strValue)       &&
         strValue!=_T("(Picture - None)") && strValue!=_T("(Image -
Aucun)"))
      {
            m_Pic.Render(pdc,Rect,Rect);
      }
      Rect.top+=Rect.Height();
      Rect.bottom+=Size.cy,
      pdc->DrawText(strCaption, Rect,DT_LEFT);
      pdc->SelectObject(pOldFont);

      if (!IsOptimizedDraw())
      {
            // Le conteneur ne prend pas en charge le dessin optimisé.

            // TODO : si vous avez sélectionné un objet GDI quelconque dans
le contexte de périphérique *pdc,
            //          restaurez ici les objets précédemment sélectionnés.
      }
}
```

VII. Ajout de l'ActiveX dans un projet graphique MFC

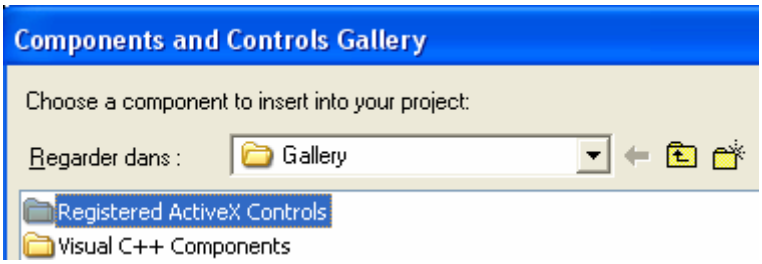
Là encore les méthodes d'insertion diffèrent avec la version de l'IDE utilisée.
Pour notre test, un simple projet boîte de dialogue fera l'affaire.
Une fois le projet généré on procédera comme suit pour insérer l'ActiveX :

VII-A. Avec Visual 6.0 :

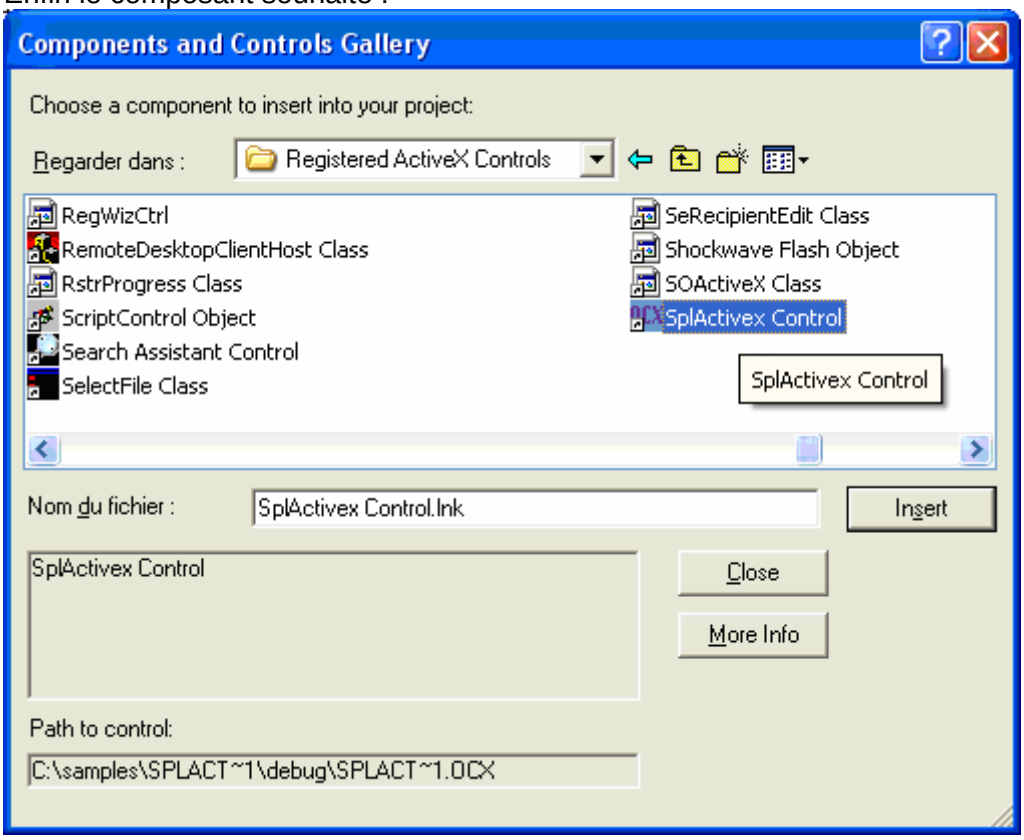
On appellera l'option **Components and Controls** :



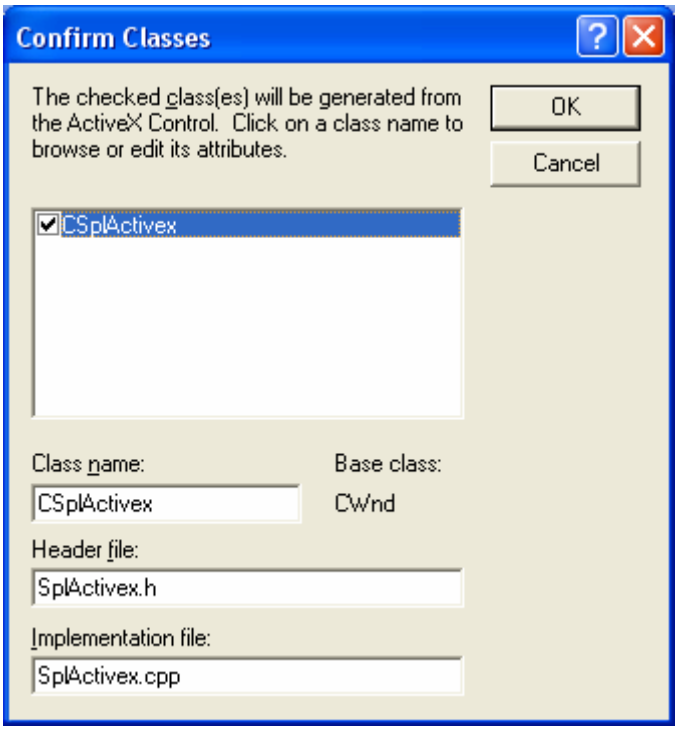
Puis on sélectionnera le répertoire **Registered ActiveX Controls**



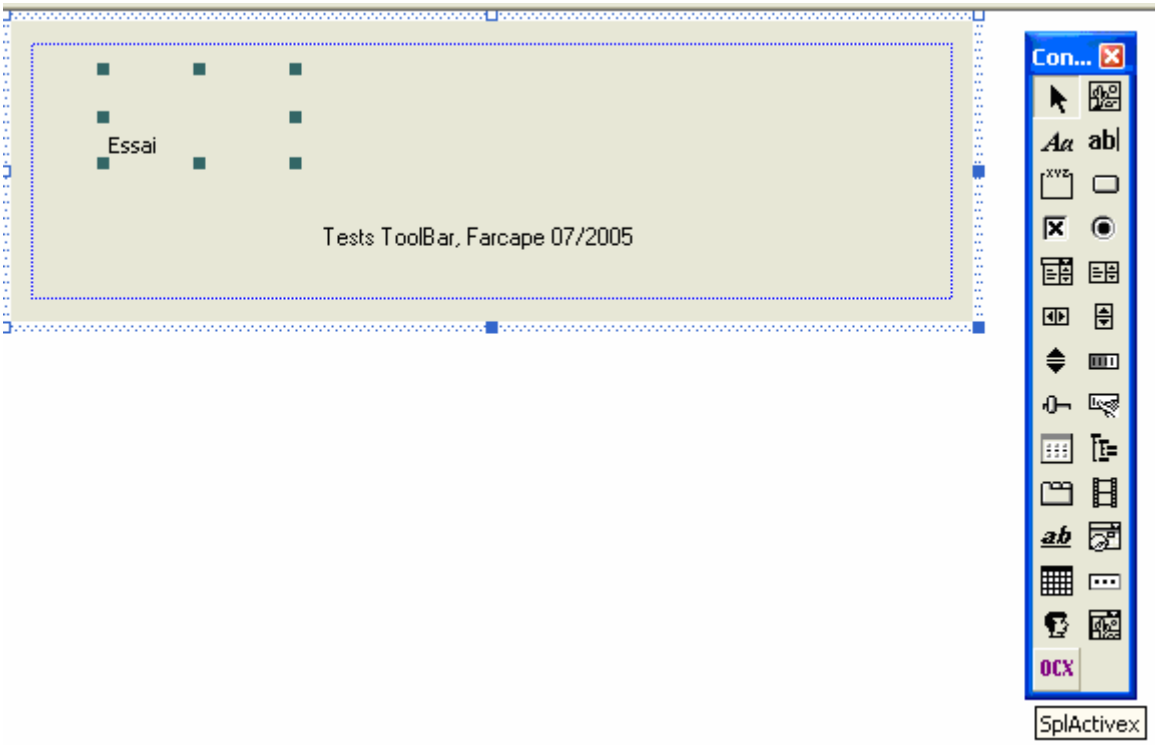
Enfin le composant souhaité :



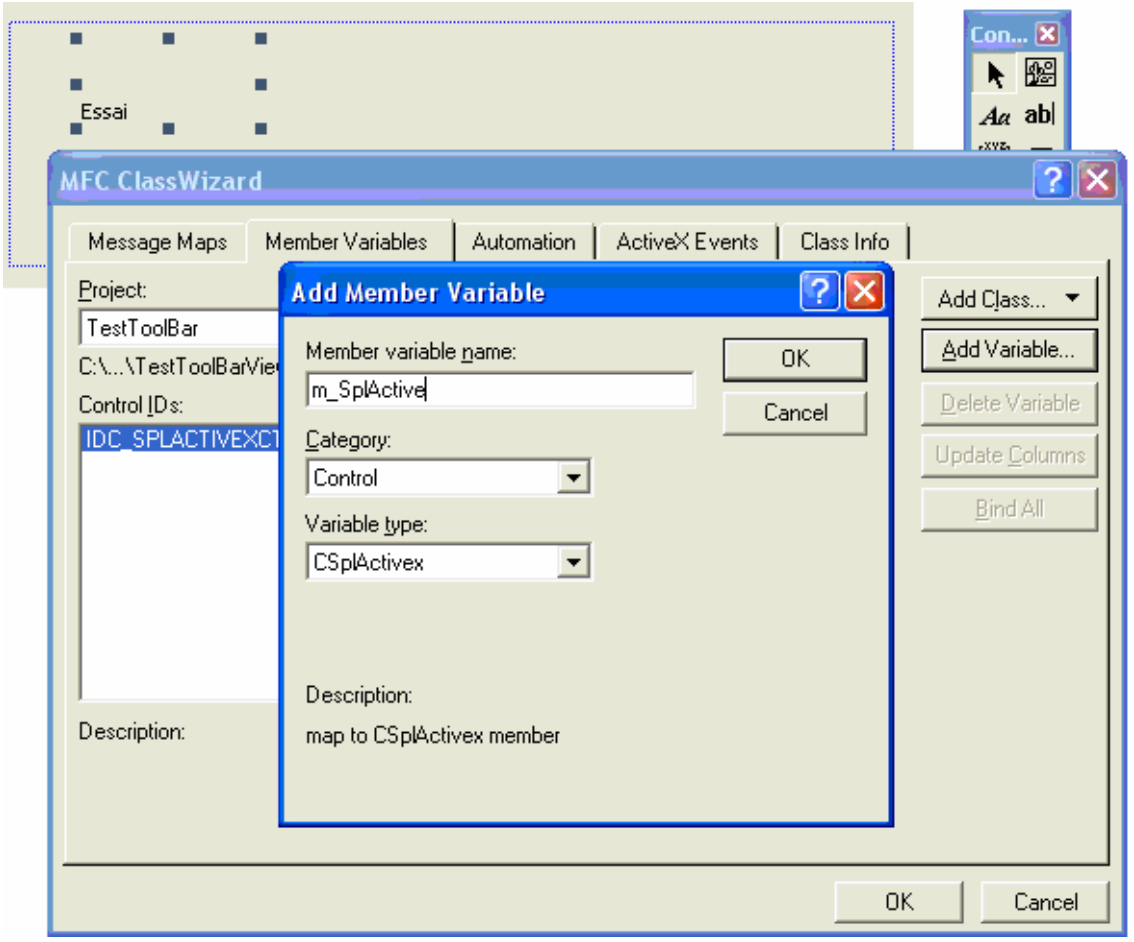
Pour insérer l'ActiveX dans la barre d'outils et générer la classe Wrapper on cliquera sur le bouton Insert :



La classe est générée et l'ActiveX est disponible dans la barre d'outils :



Enfin pour associer une variable contrôle à l'ActiveX on procédera comme pour les autres contrôles en appelant ClassWizard (CTRL+W)



VII-B. Avec Visual 2005 :

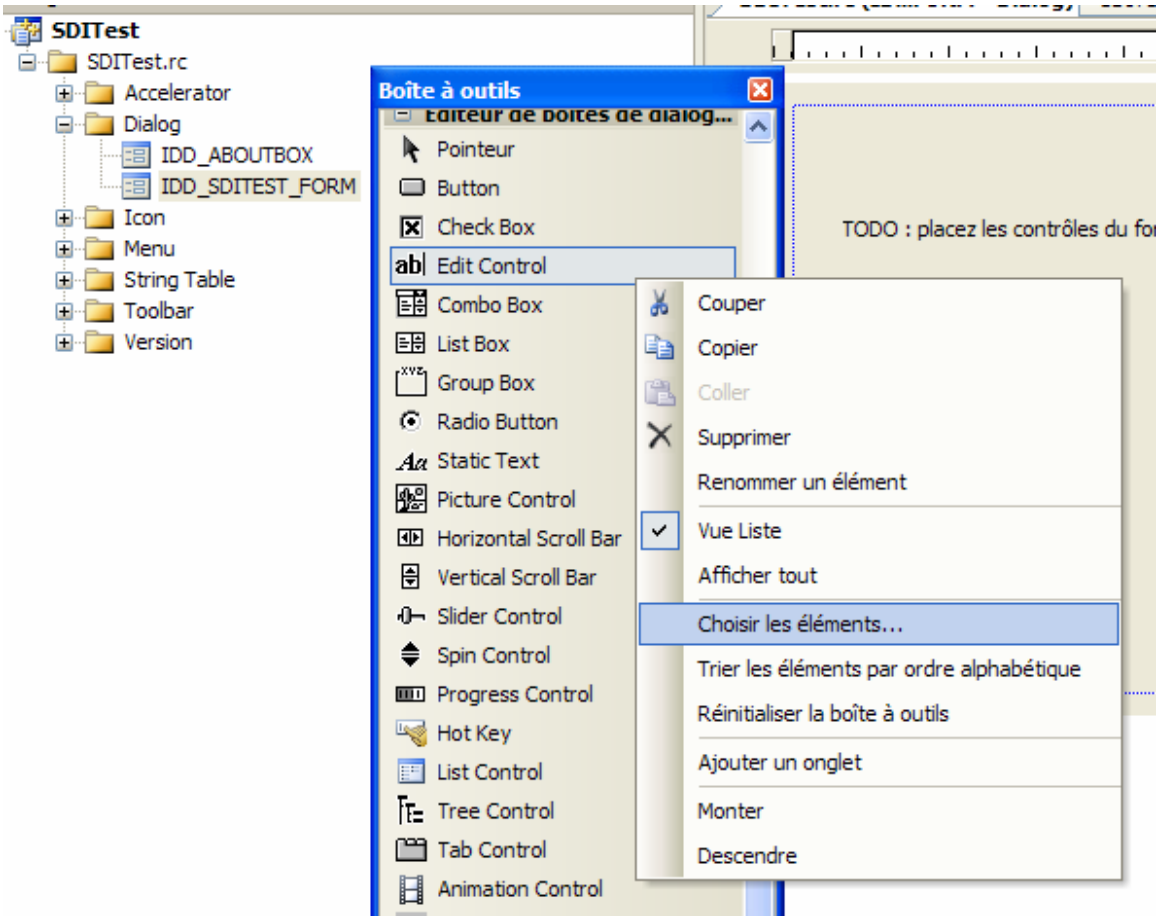
Avec **Visual 6.0** l'insertion d'un ActiveX dans un projet rajoute le composant dans la barre d'outils des ressources, et génère les classes d'interface (Wrappers) de l'ActiveX.
Avec 2005 les choses ne se passent pas tout à fait pareil.

L'insertion des ActiveX est indépendante des projets, c'est-à-dire une fois renseignés dans la barre d'outils ils sont disponibles pour tous vos projets.

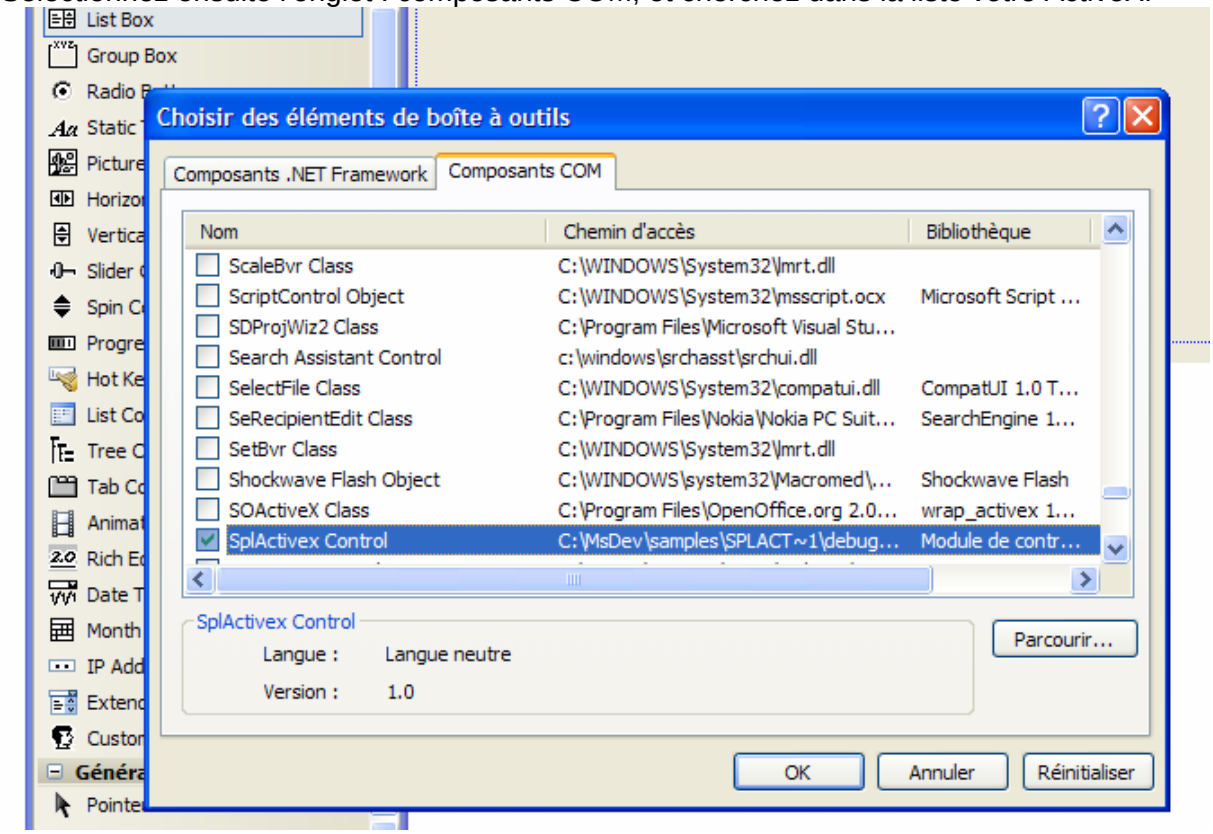
La génération des classes d'interface de l'ActiveX sera effectuée à part.
Comment procéder ?

VII-B-1. Ajout de l'ActiveX dans la barre d'outils :

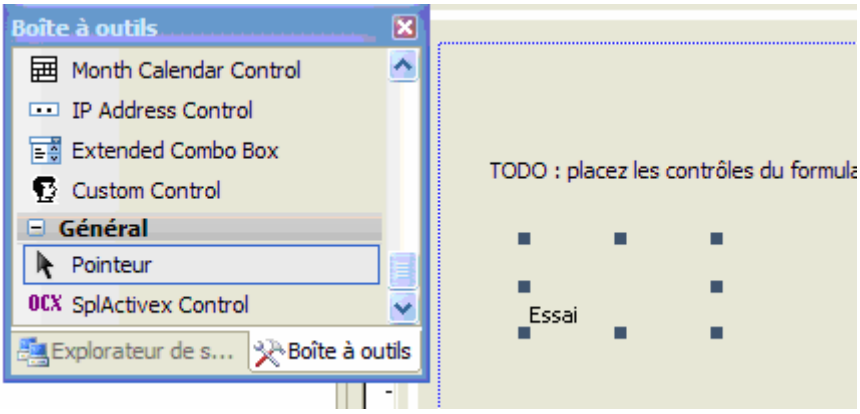
Dans le projet de test et à partir de l'éditeur de ressources et par clic droit sur la barre d'outils, sélectionnez l'option choisir un élément



Sélectionnez ensuite l'onglet : composants COM, et cherchez dans la liste votre ActiveX.



Cochez l'ActiveX et validez votre choix avec le bouton OK.

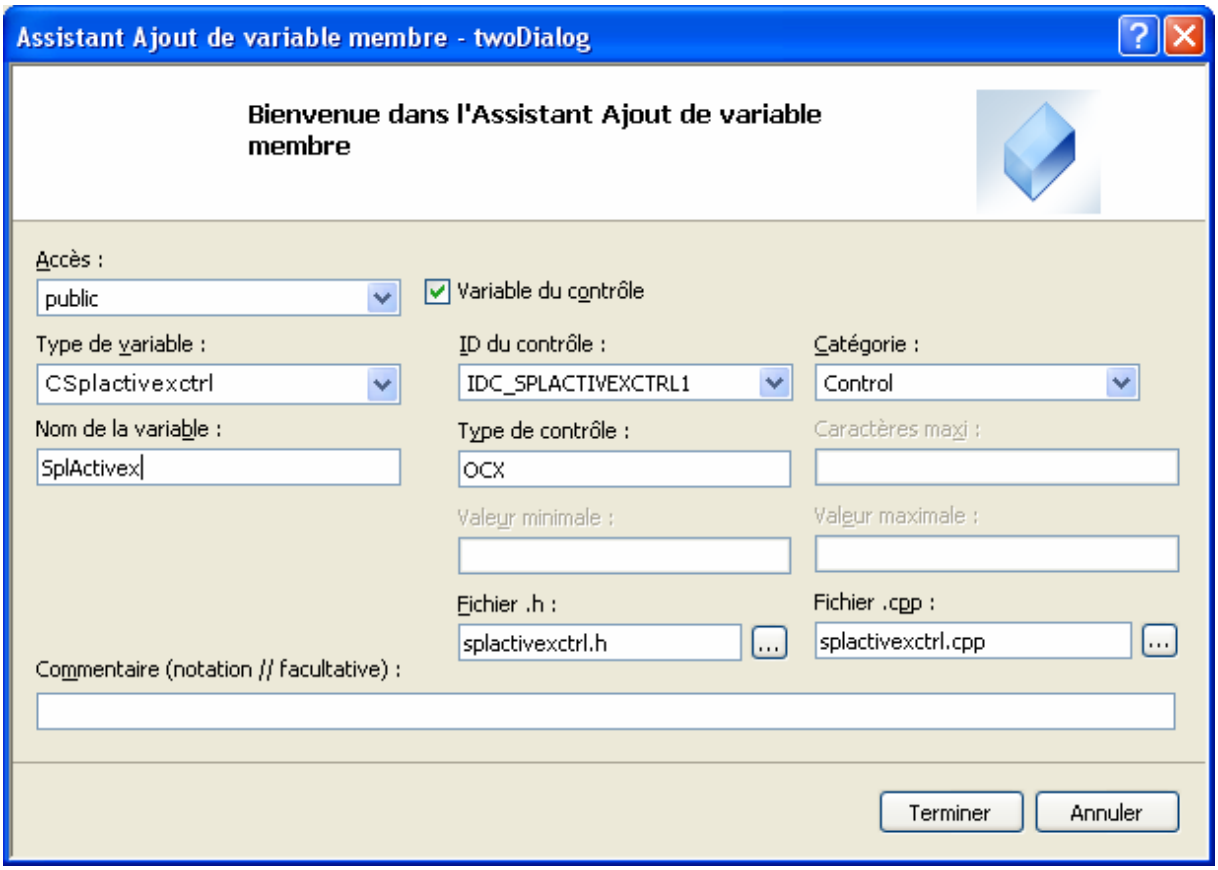


Si votre ActiveX est en cours de développement je vous déconseille néanmoins de procéder ainsi puisque l'ActiveX sera ouvert par Visual 2005 ce qui vous empêchera de le construire dans votre projet ActiveX.

On pourra alors ponctuellement ajouter un ActiveX en utilisant l'option clic droit ajouter un ActiveX sur la fenêtre des ressources.

VII-B-2. Génération de la classe Wrapper :

La génération de la classe Wrapper peut se faire au moment de la création d'une variable sur l'ActiveX.
Il suffit de faire clic droit sur l'ActiveX : Ajouter une variable.



Cliquez ensuite sur le bouton Terminer.

VII-C. La classe Wrapper :

Voici la classe Wrapper générée par l'assistant :

```
/////////////////////////////////////////////////////////////////////////////
// Classe wrapper CSplActiveXctrl

class CSplActiveXctrl : public CWnd
{
protected:
        DECLARE_DYNCREATE(CSplActiveXctrl)
public:
        CLSID const& GetClsid()
        {
        static CLSID const clsid
        = { 0x7155A63E, 0x1F44, 0x4F12, { 0xB6, 0x39, 0x38, 0xD3, 0xD6, 0x8, 0xA3, 0x9D } };
                return clsid;
        }
        virtual BOOL Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle,
                                        const RECT& rect, CWnd* pParentWnd, UINT nID,
                                        CCreateContext* pContext = NULL)
        {
        return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect, pParentWnd, nID);
        }

    BOOL Create(LPCTSTR lpszWindowName, DWORD dwStyle, const RECT& rect, CWnd* pParentWnd,
                                UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE,
                                BSTR bstrLicKey = NULL)
        {
        return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect, pParentWnd, nID,
                pPersist, bStorage, bstrLicKey);
        }
// Attributs
public:
// Opérations
public:
// _DSplActiveX

// Functions
//
        void AboutBox()
        {
                InvokeHelper(DISPID_ABOUTBOX, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
        }
        void GetLogFont(LPUNKNOWN pLogFont)
        {
                static BYTE parms[] = VTS_UNKNOWN ;
                InvokeHelper(0x3, DISPATCH_METHOD, VT_EMPTY, NULL, parms, pLogFont);
        }

// Properties
//
short GetMyVar()
{
        short result;
        GetProperty(0x1, VT_I2, (void*)&result);
        return result;
}
void SetMyVar(short propVal)
{
        SetProperty(0x1, VT_I2, propVal);
}
long GetMyWarTwo()
{
        long result;
        GetProperty(0x2, VT_I4, (void*)&result);
        return result;
}
void SetMyWarTwo(long propVal)
{
        SetProperty(0x2, VT_I4, propVal);
}
};
```

On retrouve bien les assesseurs et les modificateurs pour les variables de notre ActiveX.
Et notre méthode **GetLogFont.**

VIII. Distribution de l'ActiveX

La distribution de l'ActiveX est relativement aisée :
Il suffit de placer l'ActiveX sur le poste cible à l'emplacement de votre choix,
Et de l'enregistrer avec la commande :
Regsvr32 SplActiveX.ocx

Certains programmes comme **installshield** permettent l'enregistrement automatique de l'ActiveX au moment de l'installation du programme, sinon on peut déléguer cette tache au programme.

Le code ci-dessous provient de la documentation MSDN concernant le programme **RegSvr32** : http://msdn2.microsoft.com/en-us/library/ms177531.aspx

J'ai modifié le code pour en faire une fonction appelable directement dans une application.

Code à rajouter dans votre classe d'application :

```
BOOL CMyApp::RegisterOcx(LPCSTR szOcxName)
{
	CString str;
	CString strRegister;
	BOOL bRet=TRUE;

	str=GetProfileString("REGISTER", szOcxName,"");
	strRegister.Format("-s %s.ocx",szOcxName);
	if(str.IsEmpty())
	{
		bRet=(!::RegisterOcx(strRegister.GetBuffer(0)));
		if(bRet) WriteProfileString("REGISTER",szOcxName,"1");
	}
	return bRet;
}
```

Code à rajouter dans un fichier source nommé register.cpp.

```
#include "stdafx.h"
#include <ole2.h>
#include <stdio.h>
#include "resource.h"

#define FAIL_ARGS   1
#define FAIL_OLE    2
#define FAIL_LOAD   3
#define FAIL_ENTRY  4
#define FAIL_REG    5

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

const TCHAR _szAppName[] = _T("RegSvr32");
const char _szDllRegSvr[] = "DllRegisterServer";
const char _szDllUnregSvr[] = "DllUnregisterServer";
HINSTANCE _hInstance;
```

```
BOOL _bSilent;
BOOL _bConsole;

#define SafePutts(string) ((stdout >= 0) ? _putts(string) : 0)

void
FormatString2(
	LPTSTR lpszOut,
	LPCTSTR lpszFormat,
	LPCTSTR lpsz1,
	LPCTSTR lpsz2
	)
{
	LPCTSTR pchSrc = lpszFormat;
	LPTSTR pchDest = lpszOut;
	while (*pchSrc != '\0') {
		if (pchSrc[0] == '%' && (pchSrc[1] >= '1' && pchSrc[1] <= '2')) {
			lstrcpy(pchDest, (LPCTSTR)(pchSrc[1] == '1' ? lpsz1 : lpsz2));
			pchDest += lstrlen(pchDest);
			pchSrc += 2;
		} else {
			if (_istlead(*pchSrc))
				*pchDest++ = *pchSrc++; // copy first of 2 bytes
			*pchDest++ = *pchSrc++;
		}
	}
	*pchDest = '\0';
}

#define MAX_STRING 1024

void
DisplayMessage(
	UINT ids,
	LPCTSTR pszArg1 = NULL,
	LPCTSTR pszArg2 = NULL,
	BOOL bUsage = FALSE,
	BOOL bInfo = FALSE
	)
{
	if (_bSilent && !_bConsole)
		return;

	TCHAR szFmt[MAX_STRING];
	LoadString(_hInstance, ids, szFmt, MAX_STRING);

	TCHAR szText[MAX_STRING];
	FormatString2(szText, szFmt, pszArg1, pszArg2);
	if (bUsage) {
		int cch = _tcslen(szText);
		LoadString(_hInstance, IDS_USAGE, szText + cch, MAX_STRING - cch);
	}

	if (! _bSilent)
		MessageBox(NULL, szText, _szAppName,
			MB_TASKMODAL | (bInfo ? MB_ICONINFORMATION : MB_ICONEXCLAMATION));

	if (_bConsole) {
		TCHAR szMessage[MAX_STRING];
		FormatString2(szMessage, _T("%1: %2\n"), _szAppName, szText);
		SafePutts(szMessage);
	}
}

inline void
Usage(
```

```
	UINT ids,
	LPCTSTR pszArg1 = NULL,
	LPCTSTR pszArg2 = NULL
	)
{
	DisplayMessage(ids, pszArg1, pszArg2, TRUE);
}

inline void
Info(
	UINT ids,
	LPCTSTR pszArg1 = NULL,
	LPCTSTR pszArg2 = NULL
	)
{
	DisplayMessage(ids, pszArg1, pszArg2, FALSE, TRUE);
}
int RegisterOcx(LPSTR strarg)
{
	int iReturn = 0;
	HRESULT (STDAPICALLTYPE * lpDllEntryPoint)(void);

	BOOL bVisualC = FALSE;
	BOOL bUnregister = FALSE;
	LPCSTR pszDllEntryPoint = _szDllRegSvr;
	LPCSTR pszDllName = NULL;
	LPCSTR pszTok;
	CStringArray Listarg;
	_hInstance = AfxGetInstanceHandle();

	// Parse command line arguments.
	int iTok=0;
	while(pszTok=strtok((iTok?NULL:strarg)," "))
	{
		iTok++;
		if ((pszTok[0] == '-') || (pszTok[0] == '/'))
		{
			switch (pszTok[1])
			{
				case 'v':
				case 'V':
					bVisualC = TRUE;
					break;

				case 's':
				case 'S':
					_bSilent = TRUE;
					break;

				case 'u':
				case 'U':
					bUnregister = TRUE;
					pszDllEntryPoint = _szDllUnregSvr;
					break;

				case 'c':
				case 'C':
					_bConsole = TRUE;
					break;

				default:
					Usage(IDS_UNRECOGNIZEDFLAG, pszTok);
					return FAIL_ARGS;
			}
		}
		else
		{
```

```
                        if (pszDllName == NULL)
                        {
                                pszDllName = pszTok;
                                Listarg.Add(pszTok);
                                while((pszTok=strtok(NULL," "))) Listarg.Add(pszTok);
                                break;
                        }
                        else
                        {
                                Usage(IDS_EXTRAARGUMENT, pszTok);
                                return FAIL_ARGS;
                        }
                }
        }

        if (pszDllName == NULL)
        {
                if (bVisualC)
                        DisplayMessage(IDS_NOPROJECT);
                else
                        Usage(IDS_NODLLNAME);

                return FAIL_ARGS;
        }

        // Initialize OLE.
        if (FAILED(OleInitialize(NULL)))
        {
                DisplayMessage(IDS_OLEINITFAILED);
                return FAIL_OLE;
        }

        SetErrorMode(SEM_FAILCRITICALERRORS);      // Make sure LoadLib fails.

        for (iTok=0; iTok <Listarg.GetSize(); iTok++)
        {
                pszDllName = Listarg.GetAt(iTok);

                // Load the library.
                HINSTANCE hLib = LoadLibraryEx(pszDllName, NULL,
LOAD_WITH_ALTERED_SEARCH_PATH);

                if (hLib < (HINSTANCE)HINSTANCE_ERROR)
                {
                        TCHAR szError[12];
                        wsprintf(szError, _T("0x%08lx"), GetLastError());
                        DisplayMessage(IDS_LOADLIBFAILED, pszDllName, szError);
                        iReturn = FAIL_LOAD;
                        goto CleanupOle;
                }

                // Find the entry point.
                (FARPROC&)lpDllEntryPoint = GetProcAddress(hLib, pszDllEntryPoint);

                if (lpDllEntryPoint == NULL)
                {
                        TCHAR szExt[_MAX_EXT];
                        _tsplitpath(pszDllName, NULL, NULL, NULL, szExt);

                        if ((_stricmp(szExt, ".dll") != 0) && (_stricmp(szExt, ".ocx") != 0))
                                DisplayMessage(IDS_NOTDLLOROCX, pszDllName, pszDllEntryPoint);
                        else
                                DisplayMessage(IDS_NOENTRYPOINT, pszDllName, pszDllEntryPoint);

                        iReturn = FAIL_ENTRY;
                        goto CleanupLibrary;
                }
```

```
                // Call the entry point.
                if(FAILED((*lpDllEntryPoint)()))
                {
                        DisplayMessage(IDS_CALLFAILED, pszDllEntryPoint, pszDllName);
                        iReturn = FAIL_REG;
                        goto CleanupLibrary;
                }

                Info(IDS_CALLSUCCEEDED, pszDllEntryPoint, pszDllName);

CleanupLibrary:
                FreeLibrary(hLib);
        }

CleanupOle:
        OleUninitialize();
        Listarg.RemoveAll();
        return iReturn;
}
```

Les chaines de caractères suivantes devront être rajoutées dans votre table de chaines dans les ressources (string table):

```
IDS_USAGE  "\n\nUsage: regsvr32 [/u] [/s] dllname\n/u - Unregister server\n/s - Silent; display no message boxes\n/c - Console output"
IDS_UNRECOGNIZEDFLAG    "Unrecognized flag: %1"
IDS_EXTRAARGUMENT       "Extra argument on command line: %1"
IDS_NOPROJECT           "This command is only valid when an OLE Custom Control project is open."
IDS_NODLLNAME           "No DLL name specified."
IDS_OLEINITFAILED       "OleInitialize failed."
IDS_LOADLIBFAILED       "LoadLibrary(""%1"") failed.\nGetLastError returns %2."
IDS_NOTDLLOROCX   "%1 was loaded, but the %2 entry point was not found.\n\n%1 does not appear to be a .DLL or .OCX file."
IDS_NOENTRYPOINT        "%1 was loaded, but the %2 entry point was not found.\n\n%2 may not be exported, or a corrupt version of %1 may be in memory. Consider using PView to detect and remove it."
IDS_CALLFAILED          "%1 in %2 failed."
IDS_CALLSUCCEEDED       "%1 in %2 succeeded."
```

Utilisation :
L'appel devra être fait dans la fonction **InitInstance** de votre application MFC utilisant l'ActiveX.

Code :

```
if(!RegisterOcx("SplActiveX"))
{
      AfxMessageBox("Erreur d'enregistrement de l'ActiveX:SplActiveX.ocx");
      return FALSE;
}
```

IX. Conclusions

En dehors de quelques différences dues à l'interface de développement la création d'un ActiveX est assez similaire sur les deux plateformes que sont Visual 6.0 et Visual studio 2005.
Néanmoins et vu les problèmes rencontrés avec Visual 2005 ma préférence de développement va à Visual 6.0 et son assistant ClassWizard qui permet d'avoir tous les éléments rassemblés en une seule interface.

La migration d'un ActiveX créé à partir de Visual 6.0 vers Visual 2005 ne cause pas de problème particulier.

Dernière remarque quand vous développez votre ActiveX surveillez bien sa construction et son enregistrement, Visual 2005 à la fâcheuse tendance à laisser ouvert l'ActiveX utilisé dans un projet, même après sa fermeture ! Ce qui est fréquent quand on passe du projet de conception de l'ActiveX à celui de test.
Ce problème nécessitera de fermer Visual Studio 2005 pour de nouveau pouvoir construire l'ActiveX.

X. Remerciements

Je remercie toute l'équipe du forum **Visual C++** pour sa relecture attentive du document.
Et plus particulièrement **nico-pyright(c)** et **bigboomshakala.**