

# Le subclassing

## But de ce tutoriel

Le but de ce tutoriel n'est pas seulement de vous faire découvrir le subclassing mais aussi de vous montrer comment on peut convertir un code VB6 utilisant le subclassing vers un code VB.Net n'utilisant pas d'artifices.

## La notion de fenêtre

Une fenêtre est tout objet que l'on voit à l'écran. Il peut s'agir d'une fenêtre classique ou de tout autre contrôle comme une zone de texte, un combo, un bouton radio, un bouton cliquable, un treeview...

Une fenêtre comporte :

- un ensemble de propriétés
- une fonction de gestion de son comportement
- un nom de classe : il permet de connaître le type de la fenêtre.

Une fenêtre se construit comme une classe en conception objet : on définit l'implémentation pour un type d'objet et ensuite, et seulement ensuite, on l'instancie pour en faire vivre autant de copies que l'on veut.

## Création d'un fenêtre

Pour créer une fenêtre, il faut une activité (un thread ou la fonction WinMain). Il faut aussi créer une classe qui porte un nom unique et qui contient les propriétés de base de la fenêtre :

- la fonction de fenêtre qui gère les fenêtres de cette classe
- le handle de l'icone de ce type de fenêtre
- le handle du curseur de ce type de fenêtre
- le handle de l'instance de l'application qui enregistre ce type de fenêtre (reçu en paramètre de WinMain)
- la couleur de fond de ce type de fenêtre
- le nom de la ressource (dans l'exécutable) contenant le menu de ce type de fenêtre
- le nom de cette classe de fenêtre

On remplit une structure WNDCLASS avec ces informations et on enregistre ce type de fenêtre avec la fonction RegisterClass(Ex). **Il ne faut enregistrer une classe de fenêtre qu'une seule fois même si vous créez plusieurs fenêtres de ce type dans une même instance d'un programme.**

Il faut ensuite instancier une fenêtre du type créé, et cela avec la fonction CreateWindow(Ex). Il faut lui passer les paramètres suivants :

- le style étendu : (support du drop de fichier, SDI, MDI, toolbox, bordures, barre de titre, transparence...)
- le type de fenêtre : le nom de la classe à instancier (par exemple :BUTTON, EDIT(textbox), STATIC(label))
- un titre (si une barre de titre existe) ou un nom
- le style : titre, menu système, bordure, popup, état...
- les coordonnées et la taille initiale

- la fenêtre parent si cette fenêtre est une fille d'un MDI ou un simple contrôle
- le handle d'un menu (chargé avec LoadMenu)
- le handle de l'instance de l'application en cours
- un éventuel paramètre à passer à la création de la fenêtre

Dès lors un handle (une référence) de cette fenêtre vous est retourné mais la fenêtre n'est pas encore visible. Les actions précédentes correspondent au Load de VB6.

Il faut ensuite afficher la fenêtre avec ShowWindow et UpdateWindow (pour lui faire dessiner son contenu).

Vous obtenez alors une belle fenêtre mais qui ne réagit à rien. Pour cela, il faut une file de message...

## Boucle et file de messages

Cela permet de donner vie à la fenêtre.

Sous Windows, les fenêtres sont gérées par une file de messages (événements) qui appelle une fonction de gestion. Ces messages indiquent, par exemple, des clics, déplacements, ajouts d'items mais aussi rafraîchissement, redessin...A chaque événement correspond un message. Les messages sont classés dans plusieurs catégories : système, fenêtre générale, listbox, toolbox, ... et messages personnels.

A chaque activité (thread) est associé une file de message automatiquement par le système.

Une boucle de message typique pour une fenêtre fait les actions suivantes :

- tant que l'on peut lire un message avec GetMessage
  - on traduit les éventuels messages d'entrées de caractères du code virtuel en code ascii.
  - on appelle la fonction de fenêtre de cette fenêtre pour lui faire traiter le message

## La gestion des fenêtres : les fonctions de fenêtre

Les fenêtres sont donc gérées par une fonction ou une chaîne de fonctions. **Il y a au moins une fonction par classe de fenêtre**, c'est la fonction par défaut fournie par Windows. Elle implémente les fonctionnalités de base de l'objet fenêtre comme le dessin, le texte contenu sur le contrôle...Le chaînage de fonctions de fenêtre se fait grâce à la fonction CallWindowProc. Ceci permet un traitement en couche par raffinement successif des messages traités. Le dernier subclassing mis en place reçoit les messages en premier, le second ensuite jusqu'à la fonction par défaut sauf si l'un des subclassing ne passe pas le message aux fonctions de niveau inférieur (en n'appelant pas CallWindowProc).

Une fonction de fenêtre gère toutes les instances de la même classe de fenêtre.

**Une fonction de fenêtre ne devrait gérer qu'une seule classe de fenêtres.**

Cette fonction a le prototype suivant :

- en C

```

LRESULT CALLBACK WndProc (HWND hWnd, UINT uMessage, WPARAM
wParam, LPARAM lParam)

```

- en VB6

```

Public Function WndProc (ByVal hWnd As Long, ByVal
uMessage As Long, ByVal wParam As Long, ByVal lParam As Long) As
Long

```

- hWnd : handle de la fenêtre (ou plus précisément, l'instance de la classe de fenêtre) qui reçoit le message : utile pour les appels d'API gérant les fenêtres (SetWindowText, SetParent, GetWindowLong...)
- uMessage : message envoyé à la fenêtre : indique l'action que la fonction doit faire sur la fenêtre (WM\_SETTEXT, LVM\_GETITEM...)
- wParam : paramètre dépendant du message (peut être un pointeur ou une valeur)
- lParam : paramètre dépendant du message (peut être un pointeur ou une valeur)

## Le subclassing c'est quoi ?

Ne vous êtes vous jamais dit : « tiens, ça serait bien que mon contrôle fasse ceci ou cela en réponse à cet évènement » et là, vous cherchez dans la documentation et oh surprise, aucune propriété ne vous le permet...par contre, ce comportement peut être obtenu par la modification ou la suppression d'un évènement existant. Eh bien, le subclassing, ça sert à ça...

Le subclassing sert donc à changer le comportement d'une fenêtre (au sens général du terme : une zone de texte, un combo, un treeview...) à la réception du message (évènement). Cela permet aussi de modifier le contenu d'un message ou d'en empêcher l'acheminement à la procédure qui gère la fenêtre.

Le subclassing est donc plusieurs choses :

- le fait **d'ajouter une fonction de fenêtre dans la chaîne des fonctions de fenêtre d'une instance** (et non d'une classe) de fenêtre.
- Affecter différentes procédures à différentes instances d'une même classe de fenêtre

Il permet :

- de modifier/supprimer un comportement face à un évènement particulier
- d'activer des fonctionnalités à une fenêtre particulière

## Et VB6 dans tout ça ?

VB6 gère les fonctions de fenêtres de ses contrôles avec des comportements par défaut dépendant des propriétés de ceux-ci. Il est donc inutile de tenter de créer une fenêtre avec les APIs depuis VB pour deux raisons :

- VB6 n'aime pas les fenêtres qu'il n'a pas créé lui-même : risque de plantage
- Pourquoi faire compliqué quand on peut faire simple

## Quels messages traiter ?

Cela dépend de l'action que vous voulez faire : la documentation sur MSDN permet de voir les paramètres des différents messages afin de trouver le bon et avec les bon paramètres.

Il faudra ensuite déclarée au moins une constante pour le message dont vous avez besoin.

```
Private Const UN_MESSAGE As Long = &Hxxx&
```

## Quelques types de messages

Il existe plusieurs catégories de messages qui ont des préfixes différents (extrait) :

- WM\_xxx : n'importe quel fenêtre (contrôles, MDI, SDI...)
- BM\_xxx : simple bouton (Command)
- CB\_xxx : simple liste déroulante (combobox)
- CDM\_xxx : Common dialog
- DTM\_xxx : DateTimePicker
- EM\_xxx : simple zone de texte éditable (TextBox)
- HKM\_xxx : contrôle HotKey
- IPM\_xxx : zone de saisie d'IP
- LB\_xxx : simple liste (listbox)
- LVM\_xxx : simple ListView
- MCM\_xxx : calendrier
- PBM\_xxx : ProgressBar
- SB\_xxx : simple barre de status
- SBM\_xxx : HScrollBar ou VScrollBar
- STM\_xxx : simple zone de texte non éditable (Label)
- TB\_xxx : simple barre d'outils
- TTM\_xxx : contrôle pour InfoBulle
- TVM\_xxx : simple TreeView
- UDM\_xxx : simple bouton Up/Down

## La méthode par APIs pour VB6

### *Fonctions de mise en place et de gestion du subclassing*

Le principe est le suivant :

- on obtient le handle de la fenêtre que l'on veut subclasser
- on remplace dans les propriétés de cette fenêtre (par son handle), le pointeur vers la fonction de fenêtre actuelle par un pointeur vers notre fonction
- on conserve une copie du pointeur original pour pouvoir transmettre les messages à cette fonction (pour ne pas avoir à réimplémenter le dessin du contrôle ;) par exemple)
- cela permet de se placer en début de la chaîne des fonctions de fenêtre de cette fenêtre et donc de recevoir en premier les messages (à moins q'un autre subclassing est lieu après la mise ne place du notre)
- Dans la fonction fenêtre dont on a mis l'adresse pointeur dans les propriétés
  - on vérifie qu'il n'est pas temps de retirer le subclassing avant de faire planter l'application
  - on traite le (ou les) message(s) que l'on veut avec un Select Case ou un If par exemple
  - **on passe les messages à la fonction de fenêtre suivante pour ne pas faire planter la fenêtre.** Car si on ne passe pas les messages aux autres fonctions de de fenêtre et en

particulier à la fonction de fenêtre par défaut pour la classe, la fenêtre apparaîtra comme plantée puisqu'elle ne recevra plus l'ordre de se redessiner ou de réagir aux clics. **Si on ne veut pas passer un certain message uniquement, on s'assure de bien passer les autres.**

- **Il est aussi important que la fonction de fenêtre renvoie la bonne valeur suivant le message.** Si vous gérez vous même un message, assurez-vous de bien renvoyer la valeur adéquat, sinon renvoyez la valeur que renvoie CallWindowProc.

Pour mettre en place le subclassing, il faut un module .bas :

```
'pointeur vers la fonction de fenêtre originale pour pouvoir la restaurer le
moment voulu
Private lpOldWindowProc As Long
'handle de la fenêtre que l'on subclasse
Private hWnd As Long

'indique à SetWindowLong/GetWindowLong de définir ou renvoyer la fonction de
fenêtre
Private Const GWL_WNDPROC As Long = (-4)
'indique le message qui doit déclencher la restauration de la fonction de
fenêtre avant que le programme ne plante
Private Const WM_NCDESTROY As Long = &H82&

'appel la fonction suivante dans la chaine des fonctions de fenêtres qui gèrent
la fenêtre hWnd
Private Declare Function CallWindowProc Lib "user32" Alias "CallWindowProcA"
(ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal msg As Long, ByVal
wParam As Long, ByVal lParam As Long) As Long

'change une valeur des propriétés d'une fenêtre : ici la fonction de fenêtre
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA"
(ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

'cette fonction permet de mettre en place le subclassing d'une fenêtre par son
handle
'hWnd : handle de la fenêtre que l'on veut subclasser
Public Function Subclass(ByVal hWnd As Long) As Boolean
    'il ne faut pas subclasser une fenêtre deux fois
    If hWnd = 0 Then
        'si elle n'est pas subclassée, on change sa fonction de fenêtre avec
notre nouvelle
        lpOldWindowProc = SetWindowLong(hWnd, _
                                        GWL_WNDPROC, _
                                        AddressOf WindowProc)

        'on garde une trace du handle de la fenêtre subclassée pour savoir
que l'on l'a fait
        hWnd = hWnd
        Subclass = True
    Else
        Subclass = False
    End If
End Function

'cette fonction retire le subclassing de la fenêtre actuellement subclassée
Public Function UnSubclass() As Boolean
    'on ne retire le subclassing que s'il y est
    If hWnd <> 0 Then
        SetWindowLong(hWnd, _
                      GWL_WNDPROC, _
                      lpOldWindowProc)
```

```

        hWindow = 0
        UnSubclass = True
    Else
        UnSubclass = False
    End If
End Function

'fonction qui est appelées pour chaque message uMsg que la fenêtre reçoit
'hWnd : handle de la fenêtre qui reçoit le message
'uMsg : message reçu
'wParam et lParam : paramètres du message
Public Function WindowProc(ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam
As Long, ByVal lParam As Long)
    'enlever le subclassing à la destruction de la fenêtre
    'pour éviter le plantage de l'application genre appel de code qui n'est
    plus en mémoire...
    If msg = WM_NCDESTROY Then
        Call UnSubclass()
    End If

    'ajouter votre code de traitement des messages ici
    'avec par exemple, un Select Case avec les messages que vous gérez
    WindowProc = CallWindowProc(lpOldWindowProc, hWnd, uMsg, wParam, lParam)
End Function

```

Présenter comme ceci, ce subclassing ne fait rien. Vous devez ajouter votre code dans la fonction WindowProc sous forme d'un If ou d'un Select :

```

    If uMsg = UN_MESSAGE Then
        'traitement
    End If

    'ou

    Select uMsg
        Case UN_MESSAGE
            'traitement
        Case UN_AUTRE_MESSAGE
            'un autre traitement
    End Select

```

Il est important que *WindowProc* renvoie une valeur en fonction du message s'il le traite. Si vous ne voulez pas passer le message plus bas dans la chaîne de procédure, affectez une valeur à WindowProc et faites un *Exit Function*.

**ATTENTION : il est vitable pour votre programme que tout message non traité par votre subclassing soit passé aux fonctions de fenêtre de plus bas niveau que la votre sans quoi votre contrôle cesserait de répondre et ne se rafraîchirait plus.**

**ATTENTION : NE JAMAIS ARRETER L'EXECUTION DE VOTRE PROJET VB6 PAR LE BOUTON « STOP » (de la barre d'outils ou du menu Exécution) DE VB. SI VOUS FAITES CELA, VB S'ARRETERA BRUTALEMENT ET VOUS PERDREZ VOS MODIFICATIONS si elles n'ont pas été enregistrées avant l'exécution du projet.**

## Lancement et arrêt du subclassing

Pour lancer le subclassing d'un contrôle : `Call Subclass (objet.hWnd)`. Ce code se trouve en général dans `Form_Load`.

Pour arrêter le subclassing d'un contrôle : `Call UnSubclass (objet.hWnd)`. Ce code se trouve en général dans `Form_Unload` voire `Form_Terminate`.

### Exemple :

voir mon code sur comment empêcher le déplacement des icônes dans un treeview.

## La méthode VB.Net : subclassing par dérivation

Pour subclasser en VB.net, c'est beaucoup plus simple qu'avec VB6. Il suffit que vous dériviez une classe de *NativeWindow* du l'espace de nom *System.Windows.Forms*. Ensuite il vous suffit de prendre en paramètre du constructeur le handle de la fenêtre à subclasser : *ByVal handle As IntPtr*. Le constructeur appelle ensuite la méthode *AssignHandle* de la classe de base avec ce paramètre. Enfin, il vous faut redéfinir la fonction membre *WndProc* de *NativeWindow* pour être notifié des messages destinés à la fenêtre.

Il y a deux façons de concevoir la classe de subclassing :

- le traitement se fait dans la classe même : c'est alors une classe spécialisée
- Le traitement se fait par un évènement déclenché pour chaque message. Ceci peut s'avérer plus lent.

Voici le code de la deuxième méthode (qui peut servir de base pour la première méthode) :

```
Imports System.Windows.Forms

Public Class SubClassing
    Inherits System.Windows.Forms.NativeWindow

    'Evènement déclencher à chaque message reçu par le contrôle
    Public Event CallbackProc(ByRef m As Message)

    'indique si le contrôle est subclassé ou non
    Private m_Subclassed As Boolean = False

    'on passe le handle du contrôle que l'on veut subclassé
    'à notre classe
    Public Sub New(ByVal handle As IntPtr)
        'on assigne le handle à la classe dont on hérite
        MyBase.AssignHandle(handle)
    End Sub

    'définit ou renvoie l'état du subclassing du contrôle
    Public Property SubClass() As Boolean
        Get
            Return m_Subclassed
        End Get
        Set(ByVal Value As Boolean)
            m_Subclassed = Value
        End Set
    End Property
End Class
```

```

        End Set
    End Property

    'la fonction de traitement des messages du contrôle
    Protected Overrides Sub WndProc(ByRef m As Message)
        'à la place de ceci vous pouvez mettre
        'votre propre traitement des messages
        If m_Subclassed Then
            'on ne déclenche l'event que si on subclasse
            RaiseEvent CallbackProc(m)
        End If
        'on passe toujours le message à la fonction de base du contrôle
        MyBase.WndProc(m)
    End Sub

    'on définit explicitement le finalizer
    Protected Overrides Sub Finalize()
        MyBase.Finalize()
    End Sub
End Class

```

L'évènement *CallbackProc* est déclenché à chaque message. En traitant cet évènement, vous pouvez modifier le contenu du message. Pour cela, il vous suffit de déclarer une variable globale à la Form contenant le contrôle à subclasser (cela peut être la Form elle-même) : `Private WithEvents test As SubClassing`. Vous pouvez ainsi choisir l'évènement dans le second combo de la fenêtre de code.

Vous pouvez bien entendu traiter les message directement dans la classe `SubClassing` (ce qui vous permet de ne pas forcément passer les messages que vous traitez, à la classe de base). Je vous encourage à implémenter vos traitement de cette manière directement dans la classe. C'est plus rapide et plus encapsulé.

### Exemples :

voir mon code sur la détection de l'écran de veille.