

# Principe du Hook et utilisation d'un Hook souris

par Anthony DE DECKER ([Accueil](#))

Date de publication :

Dernière mise à jour : 02/03/2008

Ce tutoriel présente les principes généraux d'utilisation d'un Hook et illustre ces principes dans le cadre de la création d'un Hook souris global ou local.  
Il aborde également une alternative intéressante à l'utilisation d'un Hook Local.

I - Introduction.....	3
II - Avertissements.....	4
III - Principe.....	5
III-A - Définition.....	5
III-C - Fonctionnement.....	5
IV - Implémentation.....	6
IV-A - Introduction.....	6
IV-B - SetWindowsHookEx.....	6
IV-B-1 - Présentation.....	6
IV-B-2 - Paramètres.....	6
IV-B-2-a - idHook.....	6
IV-B-2-b - lpfn.....	6
IV-B-2-c - hMod.....	7
IV-B-2-d - dwThreadId.....	7
IV-B-3 - Valeur retournée.....	7
IV-B-4 - Utilisation.....	8
IV-C - CallNextHookEx.....	8
IV-C-1 - Présentation.....	8
IV-C-2 - Paramètres.....	10
IV-C-3 - Valeur retournée.....	10
IV-C-4 - Utilisation.....	10
IV-D - UnhookWindowsHookEx.....	10
IV-D-1 - Présentation.....	10
IV-D-2 - Paramètres.....	11
IV-D-3 - Valeur retournée.....	11
IV-C-4 - Utilisation.....	11
V - Alternative au Hook Local.....	12
V-A - Présentation.....	12
V-B - Exemple.....	12
VI - Hook souris.....	14
V-A - Présentation.....	14
V-B - Types de Hook souris.....	14
V-B-1 - WH_MOUSE.....	14
V-B-1 - WH_MOUSE_LL.....	14
V-C - Procédure de traitement.....	14
V-C-1 - nCode.....	14
V-C-2 - wParam.....	15
V-C-3 - lParam.....	16
V-C-3-a - Hook de type WH_MOUSE_LL.....	16
V-C-3-b - Hook de type WH_MOUSE.....	16
VII - Exemples.....	18
VII-A - Présentation.....	18
VII-B - Conception.....	18
VII-C - Développement.....	18
VII-C-1 - Création de la classe de base des Hooks souris.....	18
VII-C-2 - Création de la classe de base des Hooks souris via API.....	21
VII-C-3 - Création des classes de Hook souris.....	22
VII-C-3-a - Hook souris local.....	22
VII-C-3-b - Hook souris global.....	24
VII-C-3-c - "Hook" via PreFilter.....	25
VII-D - Utilisation.....	26
VII-E - Utilisation dans le cadre d'un Popup.....	28
VIII - Remerciements.....	31

## I - Introduction

Dans ce tutoriel, nous allons :

- Expliquer ce qu'est un Hook,
- Présenter les API nécessaires à la mise en place d'un Hook,
- Créer un Hook souris local ou global,
- Utiliser ces Hooks.

Nous verrons également une alternative à l'utilisation du Hook local et nous illustrerons tout cela via une petite application dédiée et dans le cadre de la gestion de l'affichage d'un Popup.

Ce tutoriel est basé sur l'utilisation du Framework 2.0 et du langage VB.Net

## II - Avertissements

Le code mis à disposition dans cet article se veut le plus sécurisé possible quand à l'utilisation du Hook (pose du hook seulement si besoin, levée du hook dès que possible), dans la limite des connaissances du rédacteur (évidemment). Toutefois, poser un Hook (souris ou autre) n'est jamais anodin pour l'OS et doit être réfléchi autant au niveau de sa conception que de son implémentation.

L'auteur ne saurait donc que vous conseiller de lire attentivement cet article afin de bien comprendre le code qui en découle.



***En un mot : n'utilisez le Hook qu'en dernier recours, à bon escient et en ayant bien compris ses subtilités.***

## III - Principe

### III-A - Définition

Le Hook est un "hameçon" posé par une application dans le flux de messages système d'un certain type. Il permet à cette application de "capturer" ces messages et d'y réagir.

### III-C - Fonctionnement

Techniquement, Il s'agit de positionner une procédure de traitement dans une chaîne de traitement de messages d'un type donné.

Il faut pour cela :

- Choisir le positionnement du Hook :
  - **Hook Local** : capture des messages dans le flux d'un thread (celui de l'application ou un autre),
  - **Hook Global** : capture des messages dans le flux système,
- Choisir le type de messages devant être capturés,
- Positionner le Hook au sein de la chaîne de traitement de ces messages,
- Traiter les messages par une procédure dédiée,
- Mettre à disposition les messages pour les autres Hooks de la chaîne,
- **Lorsque le Hook devient inutile, le retirer de la chaîne.**

## IV - Implémentation

### IV-A - Introduction

Nous allons présenter ici, les différentes API mises en oeuvre dans le cadre de la création d'un Hook.


### IV-B - SetWindowsHookEx

#### IV-B-1 - Présentation

Cette API permet la mise en place d'une procédure de traitement au sein d'une des chaînes de traitement de messages :

##### SetWindowsHookEx

```
<DllImport("user32")> _
Public Shared Function SetWindowsHookEx( _
    ByVal idHook As Integer, _
    ByVal lpfn As HookProc, _
    ByVal hMod As IntPtr, _
    ByVal dwThreadId As Integer) As Integer
End Function
```

 **La procédure de traitement installée par le Hook l'est toujours en début de chaîne.**  
(Cela signifie que c'est toujours la procédure de traitement installée par le dernier Hook qui est déclenchée la première).

#### IV-B-2 - Paramètres


##### IV-B-2-a - idHook

Il s'agit du type de Hook à mettre en place :

- Hook sur les messages clavier (*WH\_KEYBOARD\_LL* et *WH\_KEYBOARD*),
- Hook sur les messages souris (*WH\_MOUSE\_LL* et *WH\_MOUSE*),
- Hook sur les messages de création/destruction de fenêtres de premier plan (*WH\_SHELL*),
- etc.

##### IV-B-2-b - lpfn


C'est un pointeur sur la procédure de traitement à inscrire dans la chaîne de traitement.

 **Comme pour toute utilisation d'un délégué passé à du code non managé, l'application devra s'assurer du maintien du pointeur sur ce délégué afin d'éviter sa collecte inopinée par le Garbage Collector.**  
**Le passage d'un "AddressOf NomDeLaProcédure" est donc à proscrire, et il convient d'utiliser une fonction déléguée déclarée explicitement.**


La signature de la procédure de traitement ne dépendant pas du type de Hook, on pourra utiliser le délégué suivant :

#### Hook Procedure

```
Public Delegate Function HookProc _
  (ByVal nCode As Integer, _
  ByVal wParam As Integer, _
  ByVal lParam As IntPtr) As Integer
```

 *Les valeurs des paramètres passés à la fonction seront par contre dépendantes du type de Hook.*

Cette procédure pourra retourner une valeur différente de 0 pour indiquer au système que le message a été traité (celui-ci ne communiquera alors pas ce message à la fenêtre cible).


 ***Toutefois, elle ne devra traiter le message que si le paramètre ncode est supérieur à 0. Dans le cas contraire, elle devra retourner le résultat de l'appel à l'API CallNextHookEx (décrite plus loin).***

#### IV-B-2-c - hMod

C'est le Handle de la DLL contenant la procédure pointée par *lpfn*.

Ce paramètre peut être omis (par cela entendez, valorisé à **IntPtr.Zero**) dans le cas où cette procédure de traitement se trouve dans le même module que l'installation via **SetWindowsHookEx**.

Ceci est vrai qu'il s'agisse d'un Hook Global **ou** d'un Hook Local.

 *A titre personnel, je n'utilise donc pas ce paramètre car il me semble opportun de gérer la mise en place et la suppression du Hook au travers d'un objet levant au besoin un événement. L'ensemble du code de Hooking s'en trouve donc intégralement dans le même module.*

#### IV-B-2-d - dwThreadId

C'est l'identifiant du thread pour lequel le Hook est positionné.

Ce paramètre valorisé à 0 équivaut à positionner un Hook pour l'ensemble des threads.

**Donc :**

- Si **dwThreadId** est alimenté avec l'identifiant d'un thread, il s'agira d'un **Hook Local**, i.e. localisé sur ce thread,
- Si **dwThreadId** n'est pas défini (0), il s'agira d'un **Hook Global**, i.e. pour l'ensemble des threads.

#### IV-B-3 - Valeur retournée

Si le positionnement de la procédure de traitement au sein de la chaîne a réussi, **SetWindowsHookEx** retourne le Handle de la procédure de Hook.

En cas d'échec, la valeur retournée est 0 et le code erreur peut être obtenu via l'API **GetLastError**.

#### GetLastError

```
<DllImport("kernel32")> _
Public Shared Function GetLastError() As Integer
End Function
```

## IV-B-4 - Utilisation

Dans le cadre d'un Hook souris global, l'utilisation de **SetWindowsHookEx** peut se présenter ainsi :

### Utilisation de SetWindowsHookEx

```
Public Delegate Function HookProc(ByVal nCode As Integer, _
    ByVal wParam As Integer, _
    ByVal lParam As IntPtr) As Integer
Private Shared hMouseHook As Integer
Private Shared MouseHookProcedure As HookProc

<DllImport("user32")> _
Public Shared Function SetWindowsHookEx( _
    ByVal idHook As Integer, _
    ByVal lpfn As HookProc, _
    ByVal hMod As IntPtr, _
    ByVal dwThreadId As Integer) As Integer
End Function
<DllImport("kernel32")> _
Public Shared Function GetLastError() As Integer
End Function
#Region "Declaration constantes"
Protected Const WH_MOUSE_LL As Integer = 14
#End Region

Protected Sub StartHookingInternal()
    ' Evite le CallbackOnCollectedDelegate
    MouseHookProcedure = New HookProc(AddressOf MouseHookProc)
    hMouseHook = SetWindowsHookEx(WH_MOUSE_LL, MouseHookProcedure, _
        IntPtr.Zero, 0)
    If hMouseHook = 0 Then
        MessageBox.Show("Echec de l'installation du hook : " & GetLastError())
    Else
        MessageBox.Show("Start effectué")
    End If
End Sub

Private Function MouseHookProc(ByVal nCode As Integer, _
    ByVal wParam As Integer, _
    ByVal lParam As IntPtr) As Integer

    ' Traitement
    ' --> ici

End Function
```

## IV-C - CallNextHookEx

### IV-C-1 - Présentation

Cette API permet de passer les informations du message en cours de traitement à la procédure suivante dans la chaîne de traitement.

### CallNextHookEx

```
<DllImport("user32")> _
Public Shared Function CallNextHookEx( _
    ByVal hhk As Integer, _
    ByVal nCode As Integer, _
    ByVal wParam As Integer, _
    ByVal lParam As IntPtr) As Integer
End Function
```

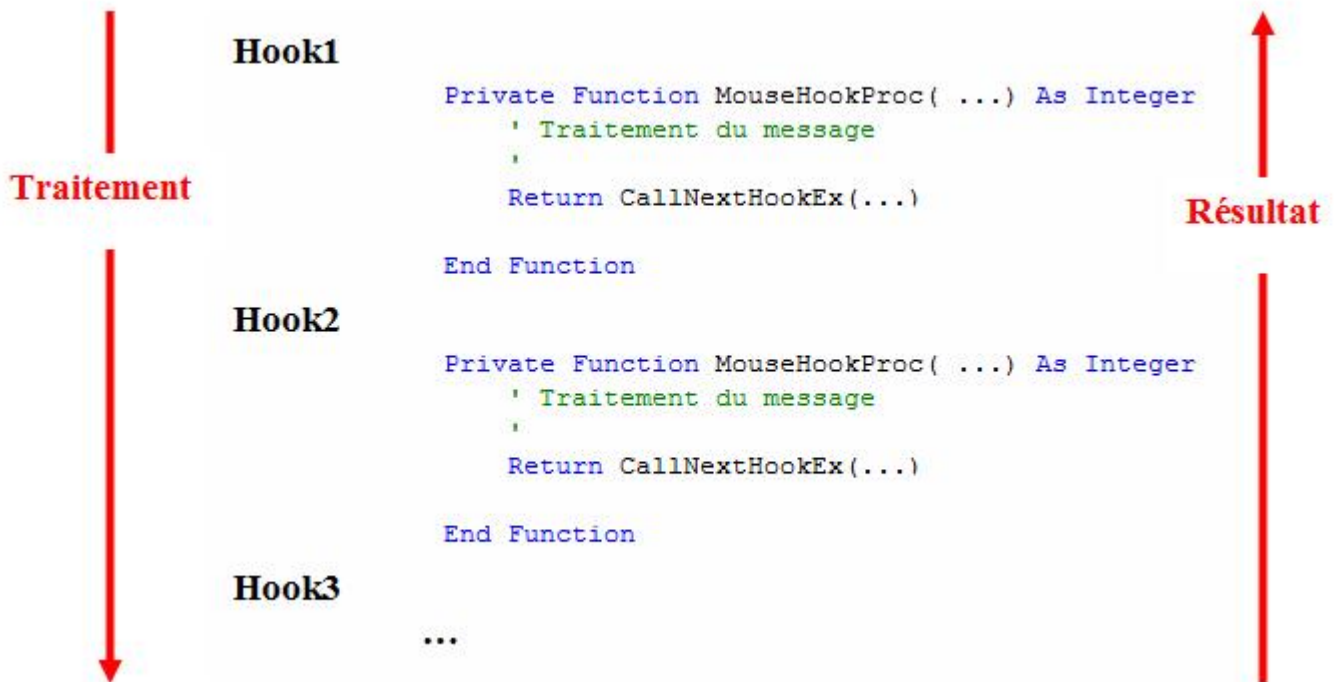
L'appel à cette fonction doit être réalisé dans la procédure de traitement positionnée par **SetWindowsHookEx** (c'est à dire dans la procédure déléguée).

Sa valeur de retour correspond à la réponse fournie par la procédure de traitement du Hook suivant de la chaîne.

**i** Une procédure de traitement retournera une valeur différente de 0 pour indiquer au système que le message a été traité et qu'il n'est pas nécessaire de le communiquer à la fenêtre cible.

En l'absence d'un blocage nécessaire du message dans la procédure de traitement de notre Hook, celle-ci doit rendre le résultat fourni par la procédure de traitement du Hook suivant de la chaîne.

Une chaîne de traitement non bloquante pour le message peut donc se représenter ainsi :



Ainsi l'utilisation de **CallNextHookEx** peut de manière générale se décliner à partir de :

#### Utilisation de CallNextHookEx

```

Public Delegate Function HookProc _
    (ByVal nCode As Integer, _
    ByVal wParam As Integer, _
    ByVal lParam As IntPtr) As Integer

Private InternalHookProcedure As HookProc
Private Shared hHook As Integer

Protected Sub StartHookingInternal()
    InternalHookProcedure = New HookProc(AddressOf InternalHookProc)
    hHook = SetWindowsHookEx( xxxxxx , MouseHookProcedure, _
        yyyy , zzzzz )
End Sub

Private Function InternalHookProc(ByVal nCode As Integer, _
    ByVal wParam As Integer, _
    ByVal lParam As IntPtr) As Integer

    ' Traitement du message ici

    Return CallNextHookEx( hHook, nCode, wParam, lParam)

End Function
    
```

**⚠ Se passer de l'appel à CallNextHookEx est fortement déconseillé.**  
**En effet, on ne sait pas dans quelle position de la chaîne se trouve notre Hook et il est donc impossible de déterminer à priori l'effet du non déclenchement des procédures de traitement positionnées en aval de celui-ci par les autres applications.**  
*Ceci sera illustré dans la suite de l'article.*

### IV-C-2 - Paramètres

Les paramètres utilisés sont ceux obtenus par la procédure de traitement auxquels s'ajoute le paramètre **hhk** qui doit être alimenté avec le Handle de la procédure de Hook courant (retourné par **SetWindowsHookEx**)

### IV-C-3 - Valeur retournée

La valeur de retour correspond à la réponse fournie par la procédure de traitement du Hook suivant de la chaîne.

### IV-C-4 - Utilisation

Dans le cadre de notre exemple pour un Hook souris global, l'utilisation de **CallNextHookEx** peut se présenter ainsi :

#### Utilisation de CallNextHookEx

```

<DllImport("user32")> _
Public Shared Function CallNextHookEx( _
    ByVal hhk As Integer, _
    ByVal nCode As Integer, _
    ByVal wParam As Integer, _
    ByVal lParam As IntPtr) As Integer
End Function
Private Function MouseHookProc(ByVal nCode As Integer,
    ByVal wParam As Integer,
    ByVal lParam As IntPtr) As Integer

    ' Traitement
    ' --> ici

    Return CallNextHookEx(hMouseHook, nCode, wParam, lParam)
End Function
    
```

## IV-D - UnhookWindowsHookEx

### IV-D-1 - Présentation

Cette API permet de retirer la procédure de traitement de la chaîne de procédures de traitement .

**⚠ Toute installation d'un Hook via SetWindowsHookEx doit être suivie d'une désinstallation via UnhookWindowsHookEx, car le Hook génère un ralentissement du traitement des messages systèmes.**  
**Les règles de "bonne conduite" à retenir sont donc :**  
 - **Installer le Hook le plus TARD possible via SetWindowsHookEx,**  
 - **Désinstaller le Hook le plus TOT possible via UnhookWindowsHookEx.**

#### UnhookWindowsHookEx

```

<DllImport("user32")> _
    
```

#### UnhookWindowsHookEx

```
Public Shared Function UnhookWindowsHookEx(ByVal idHook As Integer) As Boolean  
End Function
```

#### IV-D-2 - Paramètres

**UnhookWindowsHookEx** accepte un seul paramètre **hhk** définissant le Handle de le procédure de Hook à désinstaller (retourné par **SetWindowsHookEx**)

#### IV-D-3 - Valeur retournée

La fonction retourne *True* si la désinstallation a réussi, *False* sinon.

#### IV-C-4 - Utilisation

Dans le cadre de notre exemple pour un Hook souris global, l'utilisation de **UnhookWindowsHookEx** peut se présenter ainsi :

#### Utilisation de UnhookWindowsHookEx

```
<DllImport("user32")> _  
Public Shared Function UnhookWindowsHookEx(ByVal idHook As Integer) As Boolean  
End Function  
Protected Overrides Sub StopHookingInternal()  
    Dim blnUH As Boolean = True  
    blnUH = UnhookWindowsHookEx(hMouseHook)  
    MyBase.OnMouseLog("Stop effectué")  
End Sub
```

## V - Alternative au Hook Local

### V-A - Présentation

Une alternative intéressante à la mise en place d'un Hook Local est l'utilisation de la méthode :

#### Application.AddMessageFilter

```
Application.AddMessageFilter(value As IMessageFilter)
```

Il s'agit ici de poser un filtre sur les messages destinés à l'application. Chacun de ces messages est d'abord transmis à la méthode **PreFilterMessage** de l'objet implémentant **IMessageFilter**.

### V-B - Exemple

L'exemple suivant permet lors de la saisie d'un caractère compris entre A et Z de lever un événement avec comme paramètre le caractère en majuscule :

#### Utilisation de Application.AddMessageFilter

```
Option Explicit On
Option Strict On
Public Class FormDemarrage

    Private pf As New ExemplePreFilter

    Public Sub New()
        InitializeComponent()

        Application.AddMessageFilter(pf)
        AddHandler pf.KeyDown, AddressOf pfKeyDown
    End Sub

    Private Sub pfKeyDown(ByVal s As String)
        Me.TextBox1.Text = Me.TextBox1.Text & s
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
        Dim f As New FormDeSaisie
        f.Show()
    End Sub
End Class

Public NotInheritable Class ExemplePreFilter
    Implements Windows.Forms.IMessageFilter
    '
    Public Event KeyDown(ByVal s As String)


    Public Function PreFilterMessage(ByRef m As System.Windows.Forms.Message) _
        As Boolean Implements System.Windows.Forms.IMessageFilter.PreFilterMessage

        If m.Msg = &H100 Then ' WM_KEYDOWN
            If m.WParam.ToInt32 > 64 And m.WParam.ToInt32 < 91 Then
                RaiseEvent KeyDown(UCCase(Chr(m.WParam.ToInt32)))
            End If
        End If

        Return False
    End Function
End Class
```

Utilisation de Application.AddMessageFilter

`End Class`

 L'exemple complet est disponible ici : [Source UseApplicationPrefilter.zip](#)

## VI - Hook souris

### V-A - Présentation

Nous allons utiliser les API présentées précédemment pour créer un Hook local et un Hook global sur les messages d'activité de la souris.

Pour ce faire, nous utiliserons les types de Hook suivants :

- **WH\_MOUSE**
- **WH\_MOUSE\_LL**

### V-B - Types de Hook souris


#### V-B-1 - WH\_MOUSE

Ce type de Hook réagit aux messages fournis aux applications suite à un GetMessage (attente d'un message positionné par un application) ou un PeekMessage (vérification de la disponibilité d'un message et récupération) sur les messages d'activité de la souris.

Il peut être soit Local, soit Global.

#### V-B-1 - WH\_MOUSE\_LL

Ce type de Hook réagit aux messages fournis par le driver de la souris et générés via l'API **SendInput**. Il s'agit d'un **Hook de "bas niveau"** (d'où le LL pour Low Level).

 *De part sa nature, un Hook de "bas niveau" est toujours Global.*

### V-C - Procédure de traitement

Comme pour tout Hook, la procédure de traitement d'un Hook souris se présente ainsi :

#### Hook Procedure

```
Public Delegate Function HookProc _
    (ByVal nCode As Integer, _
    ByVal wParam As Integer, _
    ByVal lParam As IntPtr) As Integer
```

Dans le cadre d'un Hook souris, les paramètres passés à la procédure sont les suivants :

#### V-C-1 - nCode

Code destiné à déterminer quel est le traitement à effectuer.

Pour un Hook souris de type **WH\_MOUSE\_LL**, il sera toujours positionné à **HC\_ACTION** signifiant que des informations sur un message d'activité de la souris sont disponibles.

Pour un Hook de type **WH\_MOUSE**, il pourra également prendre la valeur **HC\_NOREMOVE** signifiant que des informations sur un message d'activité de la souris sont disponibles et que ce message n'a pas été supprimé de la file des messages suite à un PeekMessage avec flag **PM\_NOREMOVE**.

#### nCode


```
Protected Const HC_ACTION As Integer = 0
Protected Const HC_NOREMOVE As Integer = 3
```

## V-C-2 - wParam

Il s'agit du type de message d'activité de la souris :

#### Hook Procedure

```
' Type de messages souris
' -- Déplacement
Protected Const WM_MOUSEMOVE As Integer = &H200
' -- Bouton gauche
' ---- Zone cliente
Protected Const WM_LBUTTONDOWN As Integer = &H201
Protected Const WM_LBUTTONUP As Integer = &H202
Protected Const WM_LBUTTONDBLCLK As Integer = &H203
' ---- Zone non cliente
Protected Const WM_NCLBUTTONDOWN As Integer = &HA3
Protected Const WM_NCLBUTTONDOWN As Integer = &HA1
Protected Const WM_NCLBUTTONUP As Integer = &HA2
' -- Bouton droit
' ---- Zone cliente
Protected Const WM_RBUTTONDOWN As Integer = &H204
Protected Const WM_RBUTTONUP As Integer = &H205
Protected Const WM_RBUTTONDBLCLK As Integer = &H206
' ---- Zone non cliente
Protected Const WM_NCRBUTTONDOWN As Integer = &HA6
Protected Const WM_NCRBUTTONDOWN As Integer = &HA4
Protected Const WM_NCRBUTTONUP As Integer = &HA5
' -- Bouton central
' ---- Zone cliente
Protected Const WM_MBUTTONDOWN As Integer = &H207
Protected Const WM_MBUTTONUP As Integer = &H208
Protected Const WM_MBUTTONDBLCLK As Integer = &H209
' ---- Zone non cliente
Protected Const WM_NCMBUTTONDBLCLK As Integer = &HA9
Protected Const WM_NCMBUTTONDOWN As Integer = &HA7
Protected Const WM_NCMBUTTONUP As Integer = &HA8
' -- X Bouton
' ---- Zone cliente
Protected Const WM_XBUTTONDOWN As Integer = &H20B
Protected Const WM_XBUTTONUP As Integer = &H20C
Protected Const WM_XBUTTONDBLCLK As Integer = &H20D
' ---- Zone non cliente
Protected Const WM_NCXBUTTONDOWN As Integer = &HAB
Protected Const WM_NCXBUTTONUP As Integer = &HAC
Protected Const WM_NCXBUTTONDBLCLK As Integer = &HAD
' Molette
Protected Const WM_MOUSEWHEEL As Integer = &H20A
Protected Const WM_MOUSEHWHEEL As Integer = &H20E
```

 Un hook de type **WH\_MOUSE\_LL** ne reçoit pas de messages de type "double click" (**WM\_xxBUTTONDBLCLK**) ou "click en zone non cliente" (**WM\_NCxBUTTONxxxx**).

## V-C-3 - IParam

### V-C-3-a - Hook de type WH\_MOUSE\_LL

Dans ce cas, *IParam* contient un pointeur vers une structure **MOUSEHOOKSTRUCT** :

#### MSLLHOOKSTRUCT

```
<StructLayout(LayoutKind.Sequential)> _
Private Class MSLLHOOKSTRUCT
    Public pt As PointMHS
    Public mouseData As Integer
    Public flags As Integer
    Public time As Integer
    Public dwExtraInfo As Integer
End Class
```

Ces paramètres sont :

**- pt :**

Structure contenant les coordonnées X/Y de la position de la souris :

#### Point (API)

```
<StructLayout(LayoutKind.Sequential)> _
Protected Class PointMHS
    Public x As Integer
    Public y As Integer
End Class
```

**- mouseData :**

Dans le cas d'un message *WM\_MOUSEWHEEL*, Delta de positionnement (sur les 8 premiers bits).

**- flags :**

1 si l'événement est issu d'un SendInput (événement généré applicativement) , 0 sinon.

**- time :**

Timestamp du message : durée entre le démarrage du système et la production du message en millisecondes.

**- dwExtraInfo :**

Information supplémentaire (*non utilisée*).

### V-C-3-b - Hook de type WH\_MOUSE

Dans ce cas, *IParam* contient un pointeur vers une structure **MOUSEHOOKSTRUCTEX** :

#### MOUSEHOOKSTRUCTEX

```
<StructLayout(LayoutKind.Sequential)> _
Private Class MOUSEHOOKSTRUCTEX
    Public pt As PointMHS
    Public hwnd As Integer
    Public wHitTestCode As Integer
    Public dwExtraInfo As Integer
    Public mouseData As Integer
End Class
```

Ses paramètres sont :

**- pt :**

Structure contenant les coordonnées X/Y de la position de la souris :

**Point (API)**

```
<StructLayout(LayoutKind.Sequential)> _  
Protected Class PointMHS  
    Public x As Integer  
    Public y As Integer  
End Class
```

**- hwnd :**

Handle de la fenêtre destinatrice du message.

**- wParam :**

Code identifiant la zone de la fenêtre concernée par le message :

**Hit Test Code**

```
HTERROR = (-2)  
HTTRANSPARENT = (-1)  
HTNOWHERE = 0  
HTCLIENT = 1  
HTCAPTION = 2  
HTSYSTEMMENU = 3  
HTGROWBOX = 4  
HTSIZE = HTGROWBOX  
HTMENU = 5  
HTHSCROLL = 6  
HTVSCROLL = 7  
HTMINBUTTON = 8  
HTMAXBUTTON = 9  
HTLEFT = 10  
HTRIGHT = 11  
HTTOP = 12  
HTTOPLEFT = 13  
HTTOPRIGHT = 14  
HTBOTTOM = 15  
HTBOTTOMLEFT = 16  
HTBOTTOMRIGHT = 17  
HTBORDER = 18  
HTREDUCE = HTMINBUTTON  
HTZOOM = HTMAXBUTTON  
HTSIZEFIRST = HTLEFT  
HTSIZELAST = HTBOTTOMRIGHT  
HTOBJECT = 19  
HTCLOSE = 20  
HTHELP = 21
```

**- dwExtraInfo :**

Information supplémentaire (*non utilisée*).

**- mouseData :**

Dans le cas d'un message `WM_MOUSEWHEEL`, Delta de positionnement (sur les 8 premiers bits).

## VII - Exemples

### VII-A - Présentation

Nous allons dans cet exemple illustrer l'utilisation d'un Hook souris local, d'un Hook souris global et d'un filtre de messages.

Une interface nous permettra de mettre en évidence le fonctionnement de chaque type de Hook et nous créerons également un "trou de souris" pour chaque type de Hook (par "trou de souris", j'entends un zone qui empêche la détection de l'activité de la souris).

### VII-B - Conception

Les classes gérant nos 3 types de Hook souris (je considère ici que le Prefilter peut être vu comme une sorte de Hook), devons respecter les règles suivantes :

- Le Hook est posé le plus tard possible,
- Le Hook est retiré le plus tôt possible.

Pour ce faire, nous gérons des "abonnements" à un événement "Activité de la souris" :

**le Hook sera posé au premier abonnement et retiré lors du dernier désabonnement à l'événement.**

Comme on ne peut forcer l'utilisation du désabonnement à cet événement dans les objets qui utiliseront le Hook, nous imposerons à ces objets d'implémenter un événement informant le Hook de leur destruction.

Nos classes de gestion de Hook devront également lors de leur destruction effectuer le nettoyage nécessaire, à savoir, au minimum retirer le Hook (ou le Filtre) posé.

Dans un but d'illustration, ces classes mettront également à disposition un événement de Log qui permettra de tracer leur fonctionnement.

Les trois types de Hook ayant donc ces fonctions communes, nous allons créer une classe de base qui gèrera celles-ci.

Les Hook à base d'API nécessitant un ensemble de déclarations communes (dont les API !), nous utiliserons également une classe de base qui héritera de la classe précédente et mettra ces déclarations à disposition.

Pour finir, chacune des classes de gestion de l'un des types de Hook sera un singleton afin d'éviter, pour une même application, la création de plusieurs Hooks de même type.

C'est en effet complètement inutile dans notre exemple vu le mode de fonctionnement par abonnement à l'événement d'activité souris :

un objet devant récupérer l'activité de la souris pourra tout simplement s'abonner à l'événement sans créer une nouvelle instance de la classe gestion du Hook.

### VII-C - Développement

#### VII-C-1 - Création de la classe de base des Hooks souris

Notre classe de base par définition n'est pas instanciable (**MustInherit Class**).

Elle effectue les fonctions suivantes :

##### **1) Déclarations communes aux différents types de Hooks**

C'est l'ensemble des constantes décrivant les messages de la souris.

##### **2) Procédures communes aux différents types de Hooks**

Tout Hook, quel que soit son type, doit être posé et retiré. Nous forçons donc les classes qui héritent de notre classe de base à mettre à disposition ces procédures (en utilisant des **Protected MustOverride Sub**).

##### **3) Gestion des abonnements**

Notre classe met à disposition un événement **MouseEvent** qui sera levé à chaque détection de l'activité de la souris.

Pour gérer les abonnements à cet événement, nous utilisons un événement personnalisé (**Custom Event**) qui nous permet de respecter les règles de pose du Hook au plus tard et de retraitement au plus tôt.

L'abonnement à cet événement n'est autorisé que pour les objets implémentant **IComponent** car ceux-ci lèvent un événement **Disposed** lors de leur destruction (en l'absence de surcharge de la méthode **Dispose**). La destruction d'un objet abonné à l'événement entraîne son désabonnement d'office.

#### 4) Nettoyage

C'est l'implémentation de **IDisposable** et la gestion du retraitement du Hook lors de la destruction de l'instance.

##### Class de base des Hooks souris

```

Option Explicit On
Option Strict On
Imports System.Windows.Forms

Public Delegate Sub MouseActivityEventHandler(ByVal e As MouseEventArgs)

Public Enum eMouseHookType
    LocalMouseHook = 1
    GlobalMouseHook = 2
    PreFilterMouseHook = 3
End Enum

Public MustInherit Class CLFWAddHookBase
    Implements IDisposable

#Region "Dispose"
    Public Overloads Sub Dispose() Implements IDisposable.Dispose
        Dispose(True)
        GC.SuppressFinalize(Me)
    End Sub

    Private Disposed As Boolean = False
    Private Overloads Sub Dispose(ByVal disposing As Boolean)
        If Not Me.Disposed Then
            If disposing Then
                ' Pas de ressource managé à disposer
            End If
            RaiseEvent MouseLog("Dispose de la class Hook")
            StopHookingInternal()
            End If
            Disposed = True
            GC.SuppressFinalize(Me)
        End Sub

    Protected Overrides Sub Finalize()
        Dispose(False)
        MyBase.Finalize()
    End Sub
#End Region

#Region "Declaration constantes"
    ' Type de messages souris
    ' -- Déplacement
    Protected Const WM_MOUSEMOVE As Integer = &H200
    ' -- Bouton gauche
    ' ---- Zone cliente
    Protected Const WM_LBUTTONDOWN As Integer = &H201
    Protected Const WM_LBUTTONUP As Integer = &H202
    Protected Const WM_LBUTTONDOWNBLCLK As Integer = &H203
    ' ---- Zone non cliente
    Protected Const WM_NCLBUTTONDOWNBLCLK As Integer = &HA3
    Protected Const WM_NCLBUTTONDOWN As Integer = &HA1
    Protected Const WM_NCLBUTTONUP As Integer = &HA2
    ' -- Bouton droit
    ' ---- Zone cliente
    Protected Const WM_RBUTTONDOWN As Integer = &H204
    Protected Const WM_RBUTTONUP As Integer = &H205
    Protected Const WM_RBUTTONDOWNBLCLK As Integer = &H206
    ' ---- Zone non cliente
    Protected Const WM_NCRBUTTONDOWNBLCLK As Integer = &HA6
    Protected Const WM_NCRBUTTONDOWN As Integer = &HA4
    
```

## Class de base des Hooks souris

```

Protected Const WM_NCRBUTTONUP As Integer = &HA5
' -- Bouton central
' ---- Zone cliente
Protected Const WM_MBUTTONDOWN As Integer = &H207
Protected Const WM_MBUTTONUP As Integer = &H208
Protected Const WM_MBUTTONDOWNBLCLK As Integer = &H209
' ---- Zone non cliente
Protected Const WM_NCMBUTTONDBLCLK As Integer = &HA9
Protected Const WM_NCMBUTTONDOWN As Integer = &HA7
Protected Const WM_NCMBUTTONUP As Integer = &HA8
' -- X Bouton
' ---- Zone cliente
Protected Const WM_XBUTTONDOWN As Integer = &H20B
Protected Const WM_XBUTTONUP As Integer = &H20C
Protected Const WM_XBUTTONDOWNBLCLK As Integer = &H20D
' ---- Zone non cliente
Protected Const WM_NCXBUTTONDOWN As Integer = &HAB
Protected Const WM_NCXBUTTONUP As Integer = &HAC
Protected Const WM_NCXBUTTONDOWNBLCLK As Integer = &HAD
' Molette
Protected Const WM_MOUSEWHEEL As Integer = &H20A
Protected Const WM_MOUSEHWHEEL As Integer = &H20E

#End Region

#Region "Activité souris"

Protected MouseActivityEventHandlerList As New ArrayList
Public Custom Event MouseActivity As MouseActivityEventHandler
AddHandler(ByVal value As MouseActivityEventHandler)
    If Not MouseActivityEventHandlerList.Contains(value) Then
        ' Gestion des objets de type IComponent disposant de l'event Disposed
        If GetType(System.ComponentModel.IComponent).IsAssignableFrom( _
value.Target.GetType) Then
            Dim c As System.ComponentModel.IComponent = CType(value.Target, _
System.ComponentModel.IComponent)
            AddHandler c.Disposed, AddressOf RemoveDisposedComponent
            RaiseEvent MouseLog("Type IComponent détecté pour : " & _
value.Target.GetHashCode)
        End If

        AddMouseActivityEventHandler(value)

    End If
End AddHandler
RemoveHandler(ByVal value As MouseActivityEventHandler)
    If MouseActivityEventHandlerList.Contains(value) Then
        RemoveMouseActivityEventHandler(value)
    End If
End RemoveHandler
RaiseEvent(ByVal e As System.Windows.Forms.MouseEventArgs)
    For i As Integer = 0 To MouseActivityEventHandlerList.Count - 1
        Dim handler As MouseActivityEventHandler = _
CType(MouseActivityEventHandlerList(i), MouseActivityEventHandler)
        If handler IsNot Nothing Then
            handler.Invoke(e)
        End If
    Next
End RaiseEvent
End Event

Private Sub RemoveDisposedComponent(ByVal sender As Object, ByVal e As EventArgs)
    For i As Integer = 0 To MouseActivityEventHandlerList.Count - 1
        If CType(MouseActivityEventHandlerList(i), MouseActivityEventHandler).Target _
Is sender Then
            RemoveMouseActivityEventHandler(CType(MouseActivityEventHandlerList(i), _
MouseActivityEventHandler))
        End If
    Next
End Sub
End Sub
    
```

## Class de base des Hooks souris

```

Private Sub AddMouseActivityEventHandler(ByVal value As MouseActivityEventHandler)
    ' Première demande d'abonnement, on positionne le hook
    If MouseActivityEventHandlerList.Count = 0 Then
        RaiseEvent MouseLog("Start demandé par : " & value.Target.GetHashCode)
        StartHookingInternal()
    End If
    RaiseEvent MouseLog("Abonnement pour : " & value.Target.GetHashCode)
    MouseActivityEventHandlerList.Add(value)
End Sub
Private Sub RemoveMouseActivityEventHandler(ByVal value As MouseActivityEventHandler)
    ' Dernière demande de désabonnement, on supprime le hook
    MouseActivityEventHandlerList.Remove(value)
    RaiseEvent MouseLog("Désabonnement pour : " & value.Target.GetHashCode)
    If MouseActivityEventHandlerList.Count = 0 Then
        RaiseEvent MouseLog("Stop demandé par : " & value.Target.GetHashCode)
        StopHookingInternal()
    End If
End Sub

Protected Sub OnMouseActivity(ByVal e As MouseEventArgs)
    RaiseEvent MouseActivity(e)
End Sub

#End Region

#Region "Log"
Public Event MouseLog(ByVal Text As String)
Protected Sub OnMouseLog(ByVal Text As String)
    RaiseEvent MouseLog(Text)
End Sub
#End Region

#Region "Gestion du Hook"
Protected MustOverride Sub StartHookingInternal()
Protected MustOverride Sub StopHookingInternal()
#End Region

End Class
    
```

## VII-C-2 - Création de la classe de base des Hooks souris via API

Cette classe de base hérite de la classe de base des Hooks souris.

Elle est également par définition non instanciable (**MustInherit Class**).

Son seul but est de mutualiser l'ensemble des déclarations communes aux types de Hook souris via API : déclaration des API ainsi que des structures et constantes qu'elles utilisent.

## Class de base des Hooks souris via API

```

Option Explicit On
Option Strict On
Imports System.Runtime.InteropServices
Imports System.Windows.Forms

Public MustInherit Class CLFWAddMouseHookBase
    Inherits CLFWAddHookBase

    Public Delegate Function HookProc(ByVal nCode As Integer, _
    ByVal wParam As Integer, ByVal lParam As IntPtr) As Integer
    Protected hMouseHook As Integer
    Protected MouseHookProcedure As HookProc

    <DllImport("user32")> _
    Public Shared Function CallNextHookEx(ByVal hhk As Integer, _
    ByVal nCode As Integer, ByVal wParam As Integer, ByVal lParam As IntPtr) As Integer
    End Function
    
```

## Class de base des Hooks souris via API

```

<DllImport("user32")> _
Public Shared Function SetWindowsHookEx(ByVal idHook As Integer, _
ByVal lpfn As HookProc, ByVal hMod As IntPtr, ByVal dwThreadId As Integer) As Integer
End Function
<DllImport("user32")> _
Public Shared Function UnhookWindowsHookEx(ByVal hhk As Integer) As Boolean
End Function
<DllImport("kernel32")> _
Public Shared Function GetLastError() As Integer
End Function
<DllImport("kernel32")> _
Public Shared Function GetCurrentThreadId() As Integer
End Function

#Region "Declaration constantes"
Protected Const HC_ACTION As Integer = 0
Protected Const HC_NOREMOVE As Integer = 3
' Type de Hooks souris
Protected Const WH_MOUSE As Integer = 7
Protected Const WH_MOUSE_LL As Integer = 14
#End Region

#Region "Structures"
<StructLayout(LayoutKind.Sequential)> _
Protected Class PointMHS
    Public x As Integer
    Public y As Integer
End Class
#End Region

End Class
    
```

## VII-C-3 - Création des classes de Hook souris

Il s'agit des classes permettant l'utilisation des 3 types de Hooks souris.

## VII-C-3-a - Hook souris local

C'est une classe singleton qui hérite de la classe de base des Hooks via API. Elle gère les spécificités d'un Hook local quand à la pose et au retraitement du Hook et surtout quand à la procédure déléguée utilisée pour traiter le message (notamment l'utilisation de la structure **MOUSEHOOKSTRUCTEX**).

## Hook souris local

```

Option Explicit On
Option Strict On
Imports System.Runtime.InteropServices
Imports System.Windows.Forms

Public NotInheritable Class CLFWAddMouseHookLocal
    Inherits CLFWAddMouseHookBase

#Region "Singleton"
    Public Shared ReadOnly Property Instance() As CLFWAddMouseHookLocal
        Get
            Return Internal.instance
        End Get
    End Property

    Private Class Internal
        Friend Shared ReadOnly instance As CLFWAddMouseHookLocal = _
    End Class

    New CLFWAddMouseHookLocal
    End Class

    Private Sub New()
    
```

## Hook souris local

```

    End Sub
#End Region

#Region "Structures"
    <StructLayout(LayoutKind.Sequential)> _
    Private Class MOUSEHOOKSTRUCTEX
        Public pt As PointMHS
        Public hwnd As Integer
        Public wHitTestCode As Integer
        Public dwExtraInfo As Integer
        Public mouseData As Integer
    End Class
#End Region

Protected Overrides Sub StartHookingInternal()
    ' Evite le CallbackOnCollectedDelegate
    ' Lors du passage de délégués à du code non managé,
    ' ils doivent être maintenus actifs par l'application managée
    ' jusqu'à ce qu'il soit garanti qu'ils ne seront jamais appelés.

    MouseHookProcedure = New HookProc(AddressOf MouseHookProc)
    hMouseHook = SetWindowsHookEx(WH_MOUSE, MouseHookProcedure, _
        IntPtr.Zero, _
        GetCurrentThreadId)

    If hMouseHook = 0 Then
        MyBase.OnMouseLog("Echec de l'installation du hook : " & GetLastError())
    Else
        MyBase.OnMouseLog("Start effectué - Thread = " & _
            System.Threading.Thread.CurrentThread.ManagedThreadId)
    End If

End Sub

Protected Overrides Sub StopHookingInternal()
    Dim blnUH As Boolean = True
    blnUH = UnhookWindowsHookEx(hMouseHook)
    MyBase.OnMouseLog("Stop effectué")
End Sub

Private Function MouseHookProc(ByVal nCode As Integer, ByVal wParam As Integer, _
    ByVal lParam As IntPtr) As Integer

    If nCode = HC_ACTION Then

        Dim mbMouseButton As MouseButton = MouseButton.None
        Dim intClick As Integer

        Select Case wParam
            Case WM_LBUTTONDOWN, WM_NCLBUTTONDOWN
                mbMouseButton = MouseButton.Left
                intClick = 2
            Case WM_RBUTTONDOWN, WM_NCRBUTTONDOWN
                mbMouseButton = MouseButton.Right
                intClick = 2
            Case WM_LBUTTONUP, WM_NCLBUTTONUP
                mbMouseButton = MouseButton.Left
                intClick = 1
            Case WM_RBUTTONUP, WM_NCRBUTTONUP
                mbMouseButton = MouseButton.Right
                intClick = 1
        End Select

        If mbMouseButton <> MouseButton.None Then

            Dim mhs As MOUSEHOOKSTRUCTEX = CType(Marshal.PtrToStructure _
                (lParam, GetType(MOUSEHOOKSTRUCTEX)), MOUSEHOOKSTRUCTEX)
            Dim meaArgs As New MouseEventArgs(mbMouseButton, intClick, _
                mhs.pt.x, mhs.pt.y, 0)
            ' On ne lève l'événement que si des abonnements existent
            ' il s'agit ici de gérer l'absence d'appel à StopHooking
        End If
    End If

    Return nCode
End Function

```

## Hook souris local

```

    If MouseActivityEventHandlerList.Count > 0 Then
        MyBase.OnMouseLog("Event MouseActivity levé pour " & _
            MouseActivityEventHandlerList.Count.ToString & " abonnements")
        MyBase.OnMouseActivity(meaArgs)
    Else
        MyBase.OnMouseLog("Event MouseActivity NON levé")
    End If
End If

Return CallNextHookEx(hMouseHook, nCode, wParam, lParam)

End Function

End Class
    
```

## VII-C-3-b - Hook souris global

C'est une classe singleton qui hérite de la classe de base des Hooks via API. Elle gère les spécificités d'un Hook Global quand à la pose et au retraitement du Hook et surtout quand à la procédure déléguée utilisée pour traiter le message (notamment l'utilisation de la structure **MSLLHOOKSTRUCT** et l'absence de message de type NC).

## Hook souris Global

```

Option Explicit On
Option Strict On
Imports System.Runtime.InteropServices
Imports System.Windows.Forms

Public NotInheritable Class CLFWAddMouseHookGlobal
    Inherits CLFWAddMouseHookBase

    #Region "Singleton"
    Public Shared ReadOnly Property Instance() As CLFWAddMouseHookGlobal
        Get
            Return Internal.Instance
        End Get
    End Property

    Private Class Internal
        Friend Shared ReadOnly Instance As CLFWAddMouseHookGlobal = _
            New CLFWAddMouseHookGlobal
        End Class

    Private Sub New()
    End Sub
    #End Region

    #Region "Structures"
    <StructLayout(LayoutKind.Sequential)> _
    Private Class MSLLHOOKSTRUCT
        Public pt As PointMHS
        Public mouseData As Integer
        Public flags As Integer
        Public time As Integer
        Public dwExtraInfo As Integer
    End Class
    #End Region

    Protected Overrides Sub StartHookingInternal()
        ' Evite le CallbackOnCollectedDelegate
        ' Lors du passage de délégués à du code non managé,
        ' ils doivent être maintenus actifs par l'application managée
        ' jusqu'à ce qu'il soit garanti qu'ils ne seront jamais appelés.
        MouseHookProcedure = New HookProc(AddressOf MouseHookProc)
        hMouseHook = SetWindowsHookEx(WH_MOUSE_LL, MouseHookProcedure, _
    
```

## Hook souris Global

```

IntPtr.Zero, 0)
If hMouseHook = 0 Then
    MyBase.OnMouseLog("Echec de l'installation du hook : " & GetLastError())
Else
    MyBase.OnMouseLog("Start effectué")
End If
End Sub

Protected Overrides Sub StopHookingInternal()
    Dim blnUH As Boolean = True
    blnUH = UnhookWindowsHookEx(hMouseHook)
    MyBase.OnMouseLog("Stop effectué")
End Sub

Private Function MouseHookProc(ByVal nCode As Integer, ByVal wParam As Integer, _
ByVal lParam As IntPtr) As Integer

    If nCode = HC_ACTION Then

        Dim mbMouseButton As MouseButton = MouseButton.None

        Select Case wParam
            Case WM_LBUTTONDOWN
                mbMouseButton = MouseButton.Left
            Case WM_RBUTTONDOWN
                mbMouseButton = MouseButton.Right
        End Select

        If mbMouseButton <> MouseButton.None Then
            Dim mhs As MSLHOOKSTRUCT = CType(Marshal.PtrToStructure _
                (lParam, GetType(MSLHOOKSTRUCT)), MSLHOOKSTRUCT)
            Dim meaArgs As New MouseEventArgs(mbMouseButton, 1, mhs.pt.x, _
                mhs.pt.y, 0)
            ' On ne lève l'événement que si des abonnements existent
            ' il s'agit ici de gérer l'absence d'appel à StopHooking
            If MouseActivityEventHandlerList.Count > 0 Then
                MyBase.OnMouseLog("Event MouseActivity levé pour " & _
                    MouseActivityEventHandlerList.Count.ToString & " abonnements")
                MyBase.OnMouseActivity(meaArgs)
            Else
                MyBase.OnMouseLog("Event MouseActivity NON levé")
            End If
        End If
    End If

    Return CallNextHookEx(hMouseHook, nCode, wParam, lParam)

End Function

End Class
    
```

## VII-C-3-c - "Hook" via PreFilter

C'est une classe singleton qui hérite de la classe de base des Hooks souris (car elle n'a aucun besoin des déclarations nécessaires à l'utilisation des API).

## PreFilter

```

Option Explicit On
Option Strict On

Imports System.Windows.Forms

Public NotInheritable Class CLFWAddPreFilter
    Inherits CLFWAddHookBase
    Implements Windows.Forms.IMessageFilter
    
```

## PreFilter

```

#Region "Singleton"
    Public Shared ReadOnly Property Instance() As CLFWAddPreFilter
        Get
            Return Internal.instance
        End Get
    End Property

    Private Class Internal
        Friend Shared ReadOnly instance As CLFWAddPreFilter = New CLFWAddPreFilter
    End Class

    Private Sub New()
    End Sub
#End Region

    Public Function PreFilterMessage(ByRef m As System.Windows.Forms.Message) _
    As Boolean Implements System.Windows.Forms.IMessageFilter.PreFilterMessage

        Dim mbMouseButton As MouseButton = MouseButton.None
        Dim intClick As Integer

        Select Case m.Msg
            Case WM_LBUTTONDOWN, WM_NCLBUTTONDOWN
                mbMouseButton = MouseButton.Left
                intClick = 2
            Case WM_RBUTTONDOWN, WM_NCRBUTTONDOWN
                mbMouseButton = MouseButton.Right
                intClick = 2
            Case WM_LBUTTONDOWN, WM_NCLBUTTONDOWN
                mbMouseButton = MouseButton.Left
                intClick = 1
            Case WM_RBUTTONDOWN, WM_NCRBUTTONDOWN
                mbMouseButton = MouseButton.Right
                intClick = 1
        End Select

        If mbMouseButton <> MouseButton.None Then
            Dim meaArgs As New MouseEventArgs(mbMouseButton, intClick, _
            Control.MousePosition.X, Control.MousePosition.Y, 0)
            MyBase.OnMouseLog("Event MouseActivity levé pour " & _
            MouseEventArgsEventHandlerList.Count.ToString & " abonnements")
            MyBase.OnMouseActivity(meaArgs)
        End If

        Return False

    End Function

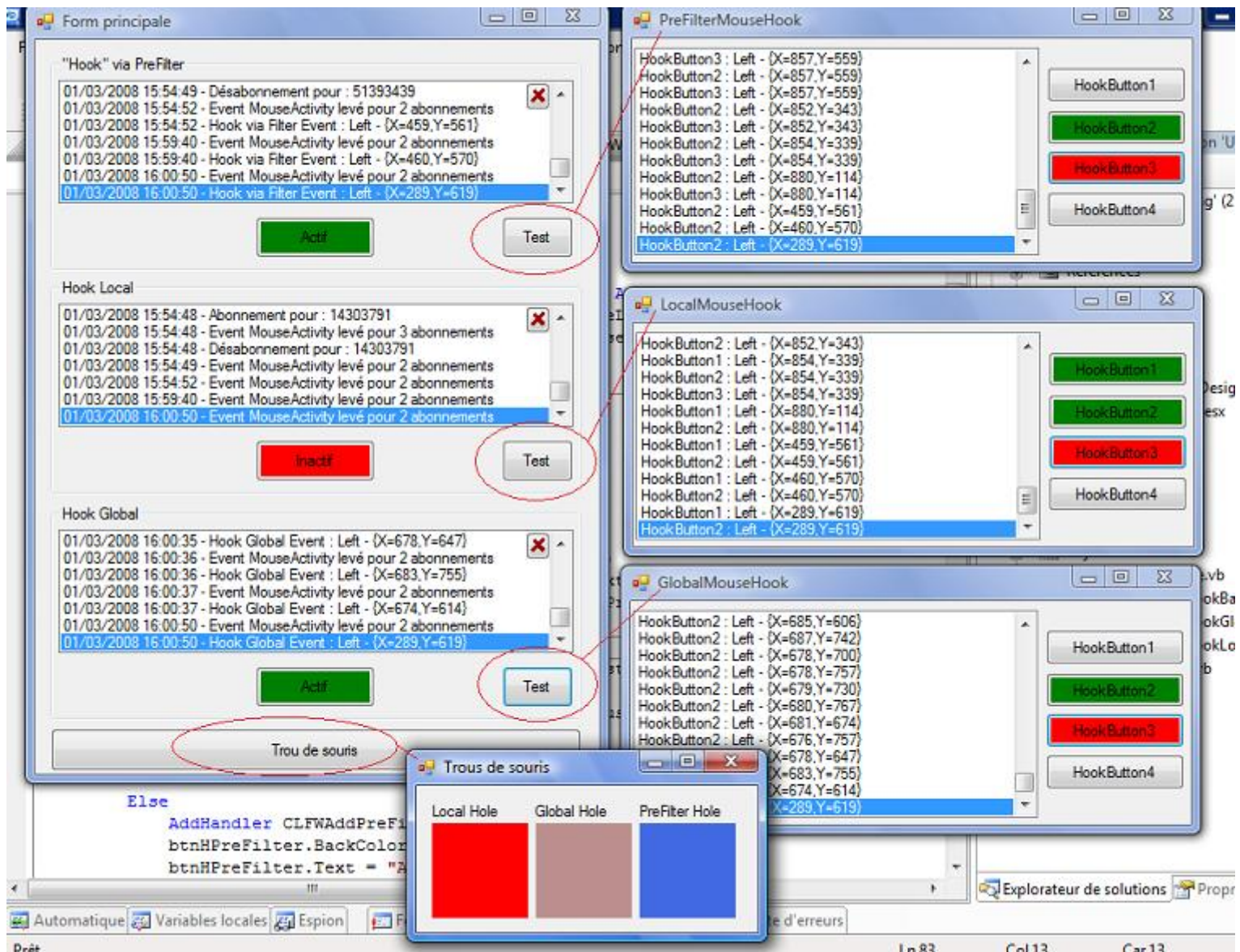
    Protected Overrides Sub StartHookingInternal()
        Application.AddMessageFilter(Me)
        MyBase.OnMouseLog("Start effectué")
    End Sub
    Protected Overrides Sub StopHookingInternal()
        Application.RemoveMessageFilter(Me)
        MyBase.OnMouseLog("Stop effectué")
    End Sub

End Class
    
```

## VII-D - Utilisation

Nous allons maintenant utiliser ces classes dans une petite application qui met en évidence leur fonctionnement. Je ne commenterai pas l'ensemble du code de cette application, mais :

 La source est disponible ici : [Source UseMouseHooking.zip](#)



Application de test des Hooks souris et du PreFilter

Au démarrage, la form principale se décompose en 4 parties :

- Un cadre dédié au Prefilter,
- Un cadre dédié au Hook Local,
- Un cadre dédié au Hook Global,
- Un bouton [trou de souris].

Chaque cadre met à disposition :

- Un bouton [Actif/Inactif] qui permet de démarrer le Hook du type concerné et de tracer l'activité de la souris remontée par ce type de Hook,
- Une liste "Log" de l'activité de la souris détectée par ce type de Hook,
- Un bouton [Test] qui déclenche l'ouverture d'un formulaire permettant de valider la gestion des abonnements/désabonnements à ce Hook (la fermeture du formulaire entraîne le désabonnement de l'ensemble des HookButtons activés par exemple).

Le bouton [Trou de souris] déclenche l'ouverture d'un formulaire "Trou de souris" composé de trois MouseHoleControl, le MouseHoleControl étant un composant personnalisé qui pose un hook local, global ou via PreFilter en bloquant le message une fois celui-ci traité :

- Pas d'appel à CallNextHookEx pour le Hook Local et le Hook Global,
- Renvoi de true (message traité) dans le PreFilter.

#### Faites quelques tests est vous constaterez que :

- Lorsque l'on ouvre le formulaire "Trou de souris" avant d'installer un Hook Local (donc via les API), le Hook du formulaire principal fonctionne correctement lorsque l'on clique sur le trou de souris "Local Hole",
- Lorsque l'on ouvre le formulaire "Trou de souris" après avoir installer un Hook Local, le Hook du formulaire principal ne fonctionne pas lorsque l'on clique sur le trou de souris "Local Hole".

Le même comportement est observé pour le Hook Global.

Ceci est normal puisque **les Hooks via API sont toujours installés en début de chaîne.**

- Lorsque l'on ouvre le formulaire "Trou de souris" avant d'installer un Prefilter, le PreFilter du formulaire principal ne fonctionne pas lorsque l'on clique sur le trou de souris "PreFilter Hole",
- Lorsque l'on ouvre le formulaire "Trou de souris" après avoir installer un Prefilter, le PreFilter du formulaire principal fonctionne correctement lorsque l'on clique sur le trou de souris "PreFilter Hole".

C'est également normal car **les filtres de messages d'application sont utilisés dans l'ordre de leur positionnement.**

## VII-E - Utilisation dans le cadre d'un Popup

Cet exemple illustre l'utilisation du Prefilter dans le cadre de la gestion de la fermeture d'un Popup.

Nous allons modifier la classe **UltraBasicPopup** présentée dans l'article précédent **Création d'un popup** pour que le Popup se ferme lors d'un clic sur un élément de l'application (mais en dehors de la **Region** du Popup).

Nous ajoutons également la gestion de la désactivation du formulaire contenant le **Control** ayant levé le Popup.

#### UltraBasicPopup avec PreFilter

```
Option Explicit On
Option Strict On
Public Class UltraBasicPopup
    Inherits Button

    #Region "Constantes"
        Private Const WS_EX_TOOLWINDOW As Integer = &H80
        Private Const SW_SHOWNOACTIVATE As Integer = 4
    #End Region

    Protected Overrides ReadOnly Property CreateParams() _
        As System.Windows.Forms.CreateParams
    Get
        Dim p As CreateParams = MyBase.CreateParams
        p.ExStyle = WS_EX_TOOLWINDOW
        p.Parent = IntPtr.Zero
        Return p
    End Get
    End Property

    Private Declare Function SetParent Lib "user32" ( _
        ByVal hWndChild As IntPtr, _
        ByVal hWndNewParent As IntPtr) As Integer
    Private Declare Function ShowWindow Lib "user32" ( _
        ByVal hWnd As IntPtr, _
        ByVal nCmdShow As Integer) As Integer
    Private Declare Function DestroyWindow Lib "user32.dll" ( _
```

## UltraBasicPopup avec PreFilter

```

        ByVal hWnd As IntPtr) As Boolean

#Region "declaration"
    Private cControlParent As Control
    Private blnOpened As Boolean
#End Region

#Region "Propriétés"
    Public ReadOnly Property Opened() As Boolean
        Get
            Return blnOpened
        End Get
    End Property
#End Region

    Private Sub ClosePopupOnDeactivate(ByVal sender As Object, ByVal e As EventArgs)
        ClosePopup()
    End Sub

    Private Sub ClosePopup()
        Dim f As Form = cControlParent.FindForm
        RemoveHandler f.Deactivate, AddressOf ClosePopupOnDeactivate
        RemoveHandler CLFWAddPreFilter.Instance.MouseActivity, _
AddressOf HPreFilterMouseActivityProc
        DestroyWindow(MyBase.Handle)
        blnOpened = False
    End Sub

    Private Sub OpenPopup()

        ' Gestion de la désactivation de la form parente du control ayant
        ' créé le popup
        Dim f As Form = cControlParent.FindForm
        AddHandler f.Deactivate, AddressOf ClosePopupOnDeactivate

        ' Gestion du clic au sein de l'application
        AddHandler CLFWAddPreFilter.Instance.MouseActivity, _
AddressOf HPreFilterMouseActivityProc
        SetParent(MyBase.Handle, IntPtr.Zero)
        ' Affichage
        ShowWindow(MyBase.Handle, SW_SHOWNOACTIVATE)
        blnOpened = True
    End Sub
    Private Sub HPreFilterMouseActivityProc(ByVal e As MouseEventArgs)
        If (Not Me.Bounds.Contains(e.Location)) Then
            ClosePopup()
        End If
    End Sub

    Public Sub ShowPopup(ByVal IControlParent As Control)

        cControlParent = IControlParent
        Me.SetLocation()
        Me.Size = New Size(150, 30)
        Me.Region = New Region( _
            New Rectangle(3, 3, Me.Size.Width - 6, Me.Size.Height - 6))
        Me.Text = "Youpi quel beau popup !"
        Me.OpenPopup()

    End Sub
    Public Sub HidePopup()

        Me.ClosePopup()

    End Sub

    Private Sub SetLocation()

        Dim rParent As Rectangle = _
            Me.cControlParent.RectangleToScreen(Me.cControlParent.ClientRectangle)
    
```

**UltraBasicPopup avec PreFilter**

```
Dim ptLocation As New Point( _
    rParent.X + rParent.Width + 10, _
    rParent.Y + rParent.Height + 10)
Me.Location = ptLocation

End Sub

Private cColor1 As Color = Color.Transparent
Private cColor2 As Color = Color.Red
Protected Overrides Sub OnClick( _
    ByVal e As EventArgs)

    MyBase.OnClick(e)


    Dim f As Form = cControlParent.FindForm
    If f Is Nothing Then Exit Sub

    Dim c As Color = f.BackColor

    If c = cColor1 Or cColor1 = Color.Transparent Then
        cColor1 = f.BackColor
        f.BackColor = cColor2
    Else
        f.BackColor = cColor1
    End If

End Sub

End Class
```

 La source complète de l'exemple est disponible ici :  
**Source** [UltraBasicPopupWithPreFilter.zip](#)

## VIII - Remerciements

Merci à **nico-pyright(c)** pour ses pistes et réponses éclairées ainsi qu'à **SaumonAgile** et **Aspic** pour leur avis sur cette article, et enfin à **comtois** pour sa relecture avisée.

Et merci à ceux qui ont pris la peine d'aller au bout de cet article, en espérant que celui-ci a répondu à leur attente.