About Fishhead Software

URL:

www.fishware.com

Email: (for fastest reponse)

support@fishware.com

Voice:

630-892-6958

Fax:

630-892-6958

Write:

Fishhead Software
912 LaFayette Street
Aurora, IL 60505

Source Code

Remember: You can not distribute the source code or use any portion of it to create commercial, shareware, or freeware ActiveX controls or similar software.

Subclassing pitfalls

- When subclassing is turned on for the control at design time, GPFs can happen. They happened because the subclassing does not expect addresses to change. But, when you make a change to the control's source code, VB changes addresses. Therefore the control will reference an invalid address.

- When running an application, always use the application's exit option. Never use VB's end button. VB's end button will not notify the controls and loaded dlls that the application has ended. The net result, memory loss and/or VB to crash.

Modifying the source code to create an OCX for your company

Please do the following to the modified control:
- Change the project name and description;
- Change the class name of the control;
- Change the OCX name;
- Change version information to reflect your company;
- Use a different load address;

Note: other people have bought the control and may sell their applications to the same client. Taking these precautions will save you and our clients many headaches.

System Requirements

The system requirements are as follows:

Windows 95 or Windows NT 4.0
Visual Basic 4.0 (32 bit) or Visual Basic 5.0 or higher
486 or higher processor
8 MB RAM
2 MB disk space

fsMsgHook ActiveX Control Version 2.00 A member of fsVBActiveX

About Fishhead Software Copyright

fsMsgHook ActiveX Control gives the developer access to the underworkings of windows, the capability to access window messaging services. fsMsgHook ActiveX control provides safe and easy access to most window messages for any window in the system.

fsMsgHook5.ocx for use with Visual Basic 4.0 (32-bit) and 5.0
GUID = 85D4255E-4618-11D2-A271-000000000000
Requires MSVBVM50.DLL

fsMsgHook6.ocx for use with Visual Basic 6.0
GUID = E7A743B2-3D69-11D2-A253-000000000000
Requires MSVBVM60.DLL

Getting Started
Frequently Asked Questions
Features and Uses

Properties
Methods
Events
Constants

System Requirements

Visit Fishhead Software on the WEB

www.fishware.com

Getting Started

fsMsgHook is easy to use. To try it out, do the following:

1. Create a new project. Select the "standard exe" project type.

2. From the Component dialog, select "Fishhead Software fsMsgHook Control for VB 5" or "Fishhead Software fsMsgHook Control for VB 6" if using Visual Basic 6. If you are using the demo version, then select "Fishhead Software fsMsgHook Control Demo".

3. Select fsMsgHook control from the tool bar and draw it on the form.

4. Add the following code to the form load event:

```
fsMsgHook1.Add Me.hWnd
```

This will tell fsMsgHook, you want to get all messages for this form.

5. Add the following code to fsMsgHook1_WndProc event:

```
Debug.Print uMsg
```

This will display each window message being sent to the current form at run time in the immediate (debug) window.

6. Goto the view menu and select to view the debug window.

7. Now run the project. Look at the debug window and move the mouse across the form. You should see many numbers being displayed in the debug window. Each number represents a message being sent to this window.

Frequently Asked Questions

Why are there two versions (fsMsgHook5.ocx and fsMsgHook6.ocx)?

fsMsgHook5.ocx was designed to be used in 32-bit Visual Basic 4 and Visual Basic 5 applications. It was created with Visual Basic 5. Therefore, fsMsgHook5.ocx requires MSVBVM50.DLL to be installed on the user's computer. Where as fsMsgHook6.ocx was designed to be used in Visual Basic 6 applications. It was created with Visual Basic 6. Therefore, fsMsgHook6.ocx requires MSVBVM60.DLL. to be installed on the users computer.

This does not mean you cannot use fsMsgHook5.ocx as part of a Visual Basic 6 application, you can. However, you will need MSVBVM60.DLL and MSVBVM50.DLL installed on the user's computer. When you run the application, both DLLs will be loaded. The Visual Basic 6 application will load MSVBVM60.DLL, while fsMsgHook5.ocx will load MSVBVM50.DLL.. This method will require more memory and will extend the load time.

How do I upgrade from fsMsgHook5.ocx or fsMsgHook6.ocx? Visual Basic gives me an error when I try to remove fsMsgHook.

To upgrade from fsMsgHook5.ocx to fsMsgHook6.ocx, you will need to open your Visual Basic project file with Notepad. Find the line that looks the one below:

```
    Object={85D4255E-4618-11D2-A271-000000000000}#1.0#0; fsMsgHook5.ocx
```

and change it to

```
    Object={E7A743B2-3D69-11D2-A253-000000000000}#1.2#0; fsMsgHook6.ocx
```

Can I use fsMsgHook6.ocx in Visual Basic 4 or 5?

Yes you can, but not recommended.

Features and Uses

Features

- Small footprint;
- Uses standard Visual Basic runtime DLL, no additional DLLs or OCXs required;
- Visual Basic 5 and higher applications can compiled OCX into the application with the source code option;
- Safe and easy to use;
- Can capture messages for any window in your application (same executable);

Uses

- To add MouseWheel capabilities to your Win95 applications;
- To change how a window looks;
- To prevent messages being sent to a window;
- Add help messages as mouse moves over other windows, controls or menus;

Copyright and Trademarks

Constants

```
Public Enum fsMHError
        fsMHErrDupWndHandle = -2147191503
        fsMHErrDupMsgHandle = -2147191502
        fsMHErrWndHandleNotFound = -2147191501
        fsMHErrMsgHandleNotFound = -2147191500
        fsMHErrNoWndHanlde =   -2147191499
        fsMHErrIndexOutofRange = -2147191498
End Enum
```

Properties

About	Standard Property
Index	Standard Property
MessageIndex	
Name	Standard Property
Object	Standard Property
Parent	Standard Property
Tag	Standard Property
WindowIndex	

MessageIndex Property

Gets or sets the index into the list of available message handles for a given window handle.

Syntax

object.Messages([hWnd As Long]).**MessageIndex** [= *integer*]

The MessageIndex property syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hwnd	A window handle. (Optional)
integer	A value indicating the default position into the list of messages.

Remarks

When set to a valid position into the list of message handles, subsequent calls to add messages will effect changes for this messages:

For example the following code will add a message to Me.hWnd:

```
fsMsgHook1.Add (Me.hWnd)                    ' Add current form to window list
fsMsgHook1.Messages().Add (WM_MOVE)         ' Add window message for current form
fsMsgHook1.Messages().MessageIndex 1        ' Will return the value for WM_MOVE
```

Note

Do not retain this value as the message handle position into the list may change.

WindowIndex Property

Gets or sets the index into the list of available window handles that are currently known by fsMsgHook control.

Syntax

object.**WindowIndex** [= *integer*]

The WindowIndex property syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
integer	A value indicating the default position into the list of windows.

Remarks

When set to a valid position into the list of windows, subsequent calls to add messages will effect changes for this window:

For example the following code will add a message to Me.hWnd:

```
fsMsgHook1.Add (Me.hWnd)                ' Add current form to window list
fsMsgHook1.Add (Command1.hWnd)          ' Add command button to window list
fsMsgHook1.WindowIndex = 1              ' Set to currrent form window
fsMsgHook1.Messages().Add (WM_MOVE)     ' Add window message for current form
```

Note

Do not retain this value as the window handle position into the list may change.

Methods

Version

Window
Add
Clear
Count
IsWindow
Item
Messages
Remove

Message
Add
Clear
Count
IsMessage
Item
Remove

Add Method (Window)

Adds a window handle to the window list.

Syntax

object.**Add** (hWnd As Long)

The Add method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hwnd	A window handle.

Remarks

This method tells fsMsgHook which windows to capture messages and return to the application through the use of the WndProc event. It will set WindowIndex to currently added window handle. If successful, it will return zero, otherwise the generated error number, a long value.

Example

```
Dim ret As Long
ret = fsMsgHook1.Add (Me.hWnd)   ' This will add the current form to the list of windows to have messages echo to.
```

Clear Method (Window)

Removes all window handles and their messages from fsMsgHook.

Syntax

object.**Clear**

The Clear method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.

Remarks

The Clear method will free fsMsgHook from processing messages. This method will remove all window handles and messages from fsMsgHook. Also, it will set Count and WindowIndex to zero. If successful, it will return zero, otherwise the generated error number, a long value.

Example

```
Dim ret As Long
ret = fsMsgHook1.Clear
```

Count Method (Window)

Returns the number of window handles for fsMsgHook.

Syntax

object.**Count**

The Count method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.

Remarks

The Count method returns the number of window handles.

Example

```
Dim n As Integer
n = fsMsgHook1.Count
```

IsWindow Method

Determines if the passed in window handle is part of the list of known window handles.

Syntax

object.**IsWindow** (hWnd As Long)

The IsWindow method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hWnd	A window handle.

Remarks

The IsWindow method will return **True** if the passed in window handle is part of the window list. Otherwise it will return **False**. Also, if True, WindowIndex will be set to the passed in window handle.

Example

```
If fsMsgHook1.IsWindow(Me.hWnd) = True Then
        ' ...
End If
```

Item Method (Window)

Returns the window handle for the given WindowIndex.

Syntax

object.**Item** ([WindowIndex As Integer])

The Item method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
WindowIndex	The index into the known window handle list. (Optional)

Remarks

The Item method will return the associated window handle for a give index. If the passed in value is missing, then the default WindowIndex will be used. If successful, it will return a window handle, otherwise zero, a long value.

Example

```
Dim hWnd As Long
hWnd = fsMsgHook1.Item (2) ' Returns the second window handle in the list.
```

Messages Method

Provides access to a window handle's associated list of messages.

Syntax

object.**Messages** ([hWnd As Long])

The Messages method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hWnd	A window handle (optional).

Remarks

The messages method for a window handle provides access to its list of messages. Through the messages method, a programmer can add and remove messages for a given window handle. If hWnd is missing, then the default WindowIndex will be used to determine which window handle messages to retrieve.

Remove Method (Window)

This method removes a window handle from the known list of window handles.

Syntax

object.**Remove** (hWnd As Long)

The Remove method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hWnd	A window handle (optional).

Remarks

Use this method to stop recieving messages for a given window. This method will remove the passed in window handle and the associated messages. If successful, returns zero, otherwise the generated error number.

Example

```
Dim ret As Long
ret = fsMsgHook1.Remove (hWnd)
```

Add Method (Message)

Adds a message handle to the message list for a given window.

Syntax

object.Messages([hWnd As Long]).**Add** (Message As Long)

The Add method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hwnd	A window handle. (Optional)
Message	A message handle.

Remarks

This method tells fsMsgHook which messages to capture for a given window. If successful, it will return zero, otherwise the generated error number, a long value.

Example

```
Dim ret As Long
ret = fsMsgHook1.Messages(Me.hWnd).Add (WM_MOVE)

'  or

Dim ret As Long
ret = fsMsgHook1.Messages().Add (WM_MOVE)
```

Clear Method (Message)

Removes all window messages for a given window handle.

Syntax

object.Messages([hWnd As Long]).**Clear**

The Clear method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hwnd	A window handle. (Optional)

Remarks

This method will remove all messages for a window handle. Also, it will set Count and MessageIndex to zero. If successful, it will return zero, otherwise the generated error number, a long value.

Example

```
Dim ret As Long
ret = fsMsgHook1.Messages(Me.hWnd).Clear

'  or

Dim ret As Long
ret = fsMsgHook1.Messages().Clear
```

Count Method (Message)

Returns the number of message handles for a given window handle.

Syntax

object.Messages([hWnd As Long]).**Count**

The Count method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hwnd	A window handle. (Optional)

Remarks

The Count method returns the number of message handles for a given window handle.

Example

```
Dim ret As Long
ret = fsMsgHook1.Messages(Me.hWnd).Count

'  or

Dim ret As Long
ret = fsMsgHook1.Messages().Count
```

IsMessage Method

Determines if the passed in message handle is part of the list of known messages for a window handle.

Syntax

object.Messages([hWnd As Long]).**IsMessage** (Message As Long)

The IsMessage method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hwnd	A window handle. (Optional)
Message	A message handle.

Remarks

The IsMessage method will return **True** if the passed in message handle is part of the message list for the given window handle. Otherwise it will return **False**. Also, if True, MessageIndex will be set to the passed in message handle.

Example

```
If fsMsgHook1.Messages(Me.hWnd).IsMessage(WM_MOVE) = True Then
        ' ...
End If

' or

If fsMsgHook1.Messages().IsMessage(WM_MOVE) = True Then
        ' ...
End If
```

Item Method (Message)

Returns the message handle for the given MessageIndex.

Syntax

object.Messages([hWnd As Long]).**Item** ([MessageIndex As Integer])

The Item method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hwnd	A window handle. (Optional)
MessageIndex	The index into the known message handle list for a given window handle. (Optional)

Remarks

The Item method will return the associated mesage handle for a give index. If the passed in value is missing, then the default MessageIndex will be used. If successful, it will return a message handle, otherwise zero, a long value.

Example

```
Dim msg As Long
msg = fsMsgHook1.Messages(hWnd).Item (3) ' Returns the third message handle in the list.

' or

Dim msg As Long
msg = fsMsgHook1.Messages(hWnd).Item () ' Returns the message handle pointed by MessageIndex.

' or

Dim msg As Long
msg = fsMsgHook1.Messages().Item () ' Returns the message handle pointed by MessageIndex and
WindowIndex.
```

Remove Method (Message)

This method removes a message handle from the known list of message handles for a given window.

Syntax

object.Messages([hWnd As Long]).**Remove** (Message As Long)

The Remove method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hwnd	A window handle. (Optional)
Message	A message handle.

Remarks

Use this method to stop receiving the passed in message handle. This method will remove the passed in message handle for the given window handle. If successful, returns zero, otherwise the generated error number.

Example

```
Dim ret As Long
ret = fsMsgHook1.Messages(Me.hWnd).Remove (WM_MOVE)

'  or

Dim ret As Long
ret = fsMsgHook1.Messages().Remove (WM_MOVE)
```

Note

Once all messages have been removed for a given window handle, the application will receive all messages for that window handle. To prevent this. just remove the window handle.

Version Method

Returns the version number for fsMsgHook.

Syntax

object.**Version** () As String

The Version method syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object;

Remarks

The Version method will return a the saved version information in the form of major.minor.revision.

Example

```
MsgBox fsMsgHook1.Version    ' Will display "1.10.0000" for the version 1.10 release
```

Events

Error
WndProc

Error Event

The Error event gets fired when an error occurs.

Syntax

Private Sub object_**Error** (ByVal nError As Long, ByVal Description As String, bCancel As Boolean)

The Error event syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.

nError	The error number being generated
	If nError > 0 then
	(*see Visual Basic help*)
	else it will be one of the following:
	fsMHErrDupWndHandle = "Duplicate window handle" = -2147191503
	fsMHErrDupMsgHandle = "Duplicate message" = -2147191502
	fsMHErrWndHandleNotFound = "Window not found" = -2147191501
	fsMHErrMsgHandleNotFound = "Message not found" = -2147191500
	fsMHErrNoWndHandleMsg = "Not a window handle" = -2147191499
	fsMHErrIndexOutofRangeMsg = "Index out of range" = -2147191498

Description	A string value representing the generated error.
bCancel	Determines if an error message dialog displays. Default is **False**.

Remarks

This event is useful if you want to centralize your error handling or prevent fsMsgHook from displaying an error (Set bCancel = True).

WndProc Event

The WndProc event gets fired whenever fsMsgHook receives a valid window message for the sub classed window.

Syntax

Private Sub object_**WndProc** (ByVal hWnd As Long, ByVal uMsg As String, ByVal wParam As Long, ByVal lParam As Long, bForward As Boolean, Result As Long)

The WndProc event syntax has these parts:

Part	Description
object	An object expression that evaluates to fsMsgHook object.
hWnd	The window handle to receive the message.
uMsg	The message being sent.
wParam	The first set of parameters.
lParam	The second set of parameters.
bForward	Determines if the message should be sent to the actual window or next in line of chain message handlers. Default is **True**.
Result	A value representing what should happen next. Default is zero.

Remarks

The WndProc can receive all or a subset of messages for a given window depending on if fsMsgHook knows which message to filter. By default, all messages will be sent for a given window. To change this behavior, you will need to determine which messages you want WndProc to receive by setting fsMsgHook1.Messages(hWnd).Add to a window message. You will need to do this for each message you want WndProc to receive. In the example below, WM_MOVE message wil be the only message sent to the WndProc event for Me.hWnd. Also, notice the use of SELECT CASE, this is the recommanded way to determine which window handle and message to process.

```
Sub Form_Load ()

fsMsgHook1.Add (Me.hWnd)
fsMsgHook1.Messages(Me.hWnd).Add WM_MOVE

End Sub

Private Sub object_WndProc (ByVal hWnd As Long, ByVal uMsg As String, ByVal wParam As Long,
ByVal lParam As Long, bForward As Boolean, Result As Long)

Select Case hWnd
      Case Me.hWnd
     Select Case uMsg
          Case WM_MOVE
               ' ... add your code ...
     End Select
End Select

End Sub
```

The settings for wParam, lParam, and Result will be different for each message. You should consult the Windows SDK or a good API book on the different settings.