Message Spy

Properties Events Example

Description

The Message Spy custom control is used to "spy" on Windows and Visual Basic messages sent or posted to Visual Basic controls and forms.

For further information on Windows messages refer to the Windows 3.1 SDK help file supplied with Visual Basic. For further information on Visual Basic messages refer to the Visual Basic API Reference help file supplied with Visual Basic.

File Name

MSGSPY.VBX

Object Type

MsgSpy

Remarks

The Message Spy custom control allows a Visual Basic application to intercept, filter or otherwise process Windows and Visual Basic messages sent or posted to Visual Basic controls and forms. This operation is known as *subclassing* and, if used carefully, can allow very powerful effects to be achieved, including functionality not otherwise provided by Visual Basic. Both pre-processing and post-processing of messages are supported.

Distribution Note When you create and distribute applications that use the Message Spy custom control, you should install the file MSGSPY.VBX in the customer's Microsoft Windows \SYSTEM subdirectory. The Visual Basic Setup Kit included with the Professional Edition provides tools to help you write setup programs that install your applications correctly.

About

This custom control was developed by Anton Software Limited as part of the **Anton Software Library/VBX**. For additional information, you can contact them at:

Anton Software Limited
40 Midfield Way
Orpington
Kent, BR5 2QJ
UK

Phone/fax: +44 (0) 81-302 4373 (24 hours)

E-mail: 100265,2172 (CompuServe)
tonys@anton.demon.co.uk (Internet)

Contact: Tony Scott

Properties

All the properties for this control are listed in the following table. Properties that apply *only* to this control are marked with an asterisk (*) and are documented in the following sections. (Note that the list order is alphabetic from top to bottom, then left to right.) See the Visual Basic *Language Reference* or Help for documentation on the remaining properties. There are no methods for this control.

*Cancel	*HiWord	*IsForm	*ReturnValue
*Control	hWnd	Left	*SpyMode
*ControlName	*hWndClear	*LoWord	Tag
Enabled	*hWndSet	*MessageName	Top
*Form	Index	*MessageNumber	
*FormName	*IsControl	Name	

hWndSet is the default value of the control.

Events

The events for this control are listed below. They apply *only* to this control and are marked with an asterisk (*) to denote that they are documented in the following sections.

*MsgProcessed *MsgReceived

Cancel Property

Description

Specifies whether the current message should be "cancelled". Valid only in the MsgReceived event.

Usage

[*form*.]*MsgSpy*.**Cancel**[= {**True** | **False**}]

Remarks

On entry to the MsgReceived event, this property will be **False.** If then set to **True**, the Windows or Visual Basic message the event was generated for will not be passed on to the subclassed control's or form's window procedure (thereby cancelling the message). In such cases, the ReturnValue property may also be set if a *non-zero* value is required to be returned to Windows or Visual Basic. Note that cancelling a message will result in the MsgProcessed event not being generated.

For example, the following code fragment ensures that mouse clicks are ignored:

```
If MsgSpy1.MessageName = "WM_LBUTTONDOWN"
    MsgSpy1.Cancel = True
ElseIf MsgSpy1.MessageName = "WM_LBUTTONDBLCLK" Then
    MsgSpy1.Cancel = True
End If
```

Data Type

Integer (Boolean)

Control, Form Properties

Description

Specify the control or form the message is intended for respectively. Valid only in the MsgReceived and MsgProcessed events.

Usage

[*form*.]*MsgSpy*.**Control**

[*form*.]*MsgSpy*.**Form**

Remarks

The IsControl and IsForm properties specify whether a message is intended for either a control or a form respectively. Either the Control or the Form property will then specify the actual control or form. All the properties and methods of the control or form will then be available. For example, the following code fragment displays the value of a subclassed form's Tag property:

```
If MsgSpy1.IsForm Then
    MsgBox MsgSpy1.Form.Tag
End If
```

The Visual Basic **TypeOf** operator may be used for controls to determine the type of the control. For example, the following code fragment adds the x-coordinate of the mouse cursor to all subclassed list boxes:

```
If MsgSpy1.IsControl Then
    If TypeOf MsgSpy1.Control Is ListBox Then
        If MsgSpy1.MessageName = "WM_MOUSEMOVE" Then
            MsgSpy1.Control.AddItem MsgSpy1.LOWORD
        End If
    End If
End If
```

Data Type

Control or Form

ControlName, FormName Properties

Description

Specify the name of the control or form the message is intended for respectively. Valid only in the MsgReceived and MsgProcessed events.

Usage

[*form*.]*MsgSpy*.**ControlName**

[*form*.]*MsgSpy*.**FormName**

Remarks

The IsControl and IsForm properties specify whether a message is intended for either a control or a form respectively. Either the ControlName or the FormName property will then specify the name of the control or form respectively. For example, the following code fragment displays the name of a subclassed control:

```
If MsgSpy1.IsControl Then
    MsgBox MsgSpy1.ControlName
End If
```

Data Type

String

HIWORD, LOWORD Properties

Description

Specify the high-order word and low-order word portions of the LParam event parameter respectively. Valid only in the MsgReceived and MsgProcessed events.

Usage

[*form*.]*MsgSpy*.**HIWORD**

[*form*.]*MsgSpy*.**LOWORD**

Remarks

These properties are the equivalents of the HIWORD and LOWORD macros available to C and C++ programmers. They are useful in a number of Windows messages in which separate data is passed in each 16-bit word of the 32-bit LParam event parameter. For example, with the WM_MOUSEMOVE message, LOWORD specifies the x-coordinate of the mouse cursor as a screen coordinate, and HIWORD the y-coordinate. The following code fragment displays the mouse cursor's position in the debug window:

```
If MsgSpy1.MessageName = "WM_MOUSEMOVE" Then
    Debug.Print MsgSpy1.LOWORD & " " & MsgSpy1.HIWORD
End If
```

Data Type

Integer

hWndSet, hWndClear Properties

Description

Specify the window handle (hWnd property) of a Visual Basic control or form that is to be subclassed, or that is no longer to be subclassed, respectively. These properties are not available at design time.

Usage

[*form*.]*MsgSpy*.**hWndSet**[= *setting*%]

[*form*.]*MsgSpy*.**hWndClear**[= *setting*%]

Remarks

Each Message Spy control is able to subclass any Visual Basic controls and/or forms within the same Visual Basic application. Each time a window handle is assigned to the hWndSet property, the corresponding control or form is subclassed. This means that the MsgReceived and MsgProcessed events will from then on be fired for each message received by the corresponding subclassed control or form. Subclassing may be removed from a subclassed control or form by setting the hWndClear property. It is not possible to subclass a Message Spy control. (Doing so will have no effect.)

The hWndSet property is the default value of the control. Therefore assigning to the control name itself will have the same effect as assigning to its hWndSet property. For example, the following code fragment uses this technique to subclass all the controls on a form (named frm), and then to subclass the form itself:

```
Dim i As Integer

For i = 0 To frm.Controls.Count - 1
      MsgSpy1 = frm.Controls(i).hWnd
Next i

MsgSpy1 = frm.hWnd
```

Data Type

Integer

IsControl, IsForm Properties

Description

Specify whether the message is for a control or a form respectively. Valid only in the MsgReceived and MsgProcessed events.

Usage

[*form*.]*MsgSpy*.**IsControl**

[*form*.]*MsgSpy*.**IsForm**

Remarks

The MsgReceived and MsgProcessed events are generated for messages received by all subclassed controls and forms. The IsControl and IsForm properties specify whether a message is intended for either a control or a form respectively.

If IsControl is **True**, IsForm will be **False**, and vice versa.

For example, the following code fragment sets a label according to whether the mouse is over a control or the form respectively:

```
If MsgSpy1.IsControl Then
    Label1 = "Control"
Else
    Label1 = "Form"
End If
```

Data Type

Integer (Boolean)

MessageName Property

Description

At run time, specifies the name of the message a MsgReceived or MsgProcessed event was generated for. Valid only in those events. At both design and run time, may be set to the message name of the message the control should respond to (valid only if the SpyMode property is set to *1 - Single Message*). The MessageNumber property will also be set accordingly.

All WM and VBM messages are supported.

Usage

[*form*.]*MsgSpy*.**MessageName**[= *setting*$]

Remarks

At run-time, this property retrieves the name of the current message. It may be used in the MsgReceived and MsgProcessed events to differentiate between different messages. There is some overhead in doing this, as the custom control must look up the message name in an internal table each time this property is accessed. For production code, and in cases when performance is important, the Msg event parameter should be used instead.

Note that at design time, the message name may be typed directly into the Property window. Alternatively, the Property window's list box may be used to select the message name.

For example, the following code fragment uses this property to detect the WM_MOUSEMOVE message:

```
If MsgSpy1.MessageName = "WM_MOUSEMOVE" Then
    ...
End If
```

The following code fragment performs the same function using the Msg event parameter:

```
Const WM_MOUSEMOVE = &H200

If Msg = WM_MOUSEMOVE Then
    ...
End If
```

The following code fragment uses this property to specify that the control should respond only to WM_KILLFOCUS messages:

```
MsgSpy1.SpyMode = 1
MsgSpy1.MessageName = "WM_KILLFOCUS"
```

Data Type

String

MessageNumber Property

Description

At run time, specifies the message number of the message a MsgReceived or MsgProcessed event was generated for. Valid only in those events. At both design and run time, may be set to the message number of the message the control should respond to (valid only if the SpyMode property is set to *1 - Single Message*). The MessageName property will also be set accordingly. (If the message number is not for a standard WM or VBM message, the MessageName property will be set to an empty string.)

Usage

[*form*.]*MsgSpy*.**MessageNumber**[= *setting*%]

Remarks

This property specifies the number of the current message. It may be used in the MsgReceived and MsgProcessed events to differentiate between different messages, although the Msg event parameter is more suited to this purpose. It may also be assigned to at both design and run time to the message number of the message the control should respond to (valid only if the SpyMode property is set to *1 - Single Message*).

For example, the following code fragment uses this property to specify that the control should respond only to WM_SETFOCUS messages:

```
Const WM_SETFOCUS = 7
MsgSpy1.SpyMode = 1
MsgSpy1.MessageNumber = WM_SETFOCUS
```

Data Type

Integer

ReturnValue Property

Description

Specifies the value to be returned to Windows or Visual Basic. Valid only in the MsgReceived and MsgProcessed events.

Usage

[*form*.]*MsgSpy*.**ReturnValue**[= *setting&*]

Remarks

On entry to the MsgReceived event, the Cancel property will be **False**. If it is then set to **True**, the message will not be passed on to the subclassed control's or form's window procedure. In such cases, the ReturnValue property may then also be set if a *non-zero* value is required to be returned to Windows or Visual Basic.

On entry to the MsgProcessed event, the ReturnValue property will be set to the value returned from the subclassed control's or form's window procedure. It may then be changed if a different value is required to be returned to Windows or Visual Basic.

For example, the following code fragment from the MsgReceived event traps WM_USER messages and ensures that a *non-zero* value is returned:

```
If MsgSpy1.MessageName = "WM_USER"
      MsgSpy1.Cancel = True
      MsgSpy1.ReturnValue = 1
End If
```

Data Type

Long

SpyMode Property

Description

Specifies whether all messages, or just a single message, should result in the MsgReceived and MsgProcessed events being generated.

Usage

[*form*.]*MsgSpy*.**SpyMode**[= *setting*%]

Remarks

Setting	Value	Description
All Messages	0	(Default) All messages should result in the MsgReceived and MsgProcessed events being generated
Single Message	1	Only the message specified by the MessageName and MessageNumber properties should result in the MsgReceived and MsgProcessed events being generated.

For example, the following code fragment specifies that all messages should be responded to:

```
MsgSpy1.SpyMode = 0
```

Data Type

Integer (Enumerated)

MsgProcessed, MsgReceived Events

Description

Occur for each message *received* by or *processed* by a subclassed Visual Basic control or form, allowing *pre-processing* and *post-processing* of messages respectively.

Syntax

Sub *MsgSpy*_**MsgReceived**(*hWnd* **As Integer**, *Msg* **As Integer**, *WParam* **As Integer**, *LParam* **As Long**)

Sub *MsgSpy*_**MsgProcessed**(*hWnd* **As Integer**, *Msg* **As Integer**, *WParam* **As Integer**, *LParam* **As Long**)

Remarks

These events allow an application to intercept, filter or otherwise process Windows and Visual Basic messages sent (or posted) to Visual Basic controls and forms. The hWnd parameter specifies the window handle of the control or form the message is or was intended for. The message number of the message itself is specified by the Msg parameter. Further message specific data may be present in the WParam and LParam parameters, depending on the message.

The MsgReceived event allows *pre-processing* of Windows and Visual Basic messages to be performed. That is, the event is generated *before* the subclassed control's or form's window procedure has received the message. Messages may then be *cancelled* by setting the Cancel property to **True** and by setting the ReturnValue property, and *modified* by changing the Msg, WParam and LParam parameters. Changing the hWnd parameter has no effect.

The MsgProcessed event allows *post-processing* of Windows and Visual Basic messages to be performed. That is, the event is generated *after* the subclassed control's or form's window procedure has received and processed the message. Messages may then be *modified* by changing the ReturnValue property, which on entry will be set to the value returned from the subclassed control's or form's window procedure.

These events will not (re)occur for any further messages that are generated directly or indirectly *during* the processing of these events, thereby preventing uncontrolled recursion.

Programmers familiar with programming for Windows using the C programming language should notice that the syntax of these events is modelled exactly on that of a normal Window procedure.

For example, the following is a complete program that allows text files to be dragged from File Manager and dropped onto its main form (or icon if minimised), and opens Notepad to edit or view the files:

```
Option Explicit

Declare Sub DragAcceptFiles Lib "shell" (ByVal hWnd As Integer, ByVal fAccept As Integer)
Declare Function DragQueryFile Lib "shell" (ByVal hDrop As Integer, ByVal iFile As Integer, ByVal
lpszFile As String, ByVal cb As Integer) As Integer
Declare Sub DragFinish Lib "shell" (ByVal hDrop As Integer)

Sub Form_Load ()
```

```
    MsgSpy1.SpyMode = 1
    MsgSpy1.MessageName = "WM_DROPFILES"
    MsgSpy1 = frmDragDrop.hWnd
    DragAcceptFiles frmDragDrop.hWnd, 1
End Sub

Sub MsgSpy1_MsgReceived (hWnd As Integer, Msg As Integer, WParam As Integer, LParam As Long)
    Dim Filename As String * 126
    Dim Result As Integer
    Result = DragQueryFile(WParam, 0, Filename, 126)
    DragFinish WParam
    Result = Shell("notepad.exe " & Filename, 1)
End Sub
```

Close Copy Print

Message Spy Example

This example illustrates how the WM_MOUSEMOVE message can be intercepted to allow a status line (in this case a 3D panel) to display information about the control the mouse cursor is currently over. In a real application this could be a short line of help text explaining the meaning of the control, a "bubble-text" window, or, if combined with speech output, a brief spoken description of the control. In this example, the control's (or form's) name and the X and Y positions of the mouse cursor relative to the control are displayed in separate 3D panels. A message is also displayed when each control receives the input focus.

```
Sub Form_Load ()
	Dim i As Integer
	For i = 0 To controls.Count - 1
		MsgSpy1 = controls(i).hWnd
	Next i
	MsgSpy1 = frm.hWnd
End Sub

Sub MsgSpy1_MsgReceived (hWnd As Integer, Msg As Integer, WParam As Integer, LParam As Long)
	Dim WindowName As String

	Dim MessageName As String
	MessageName = MsgSpy1.MessageName

	If MsgSpy1.IsControl Then
		WindowName = MsgSpy1.ControlName

		Select Case MessageName
			Case "WM_SETFOCUS"
				pnl3dMessage = WindowName + " got focus"

			Case "WM_MOUSEMOVE"
				pnl3dMessage = "Mouse is over " & WindowName
				pnl3dXPos = Str$(MsgSpy1.LOWORD)
				pnl3dYPos = Str$(MsgSpy1.HIWORD)
		End Select
	Else
		WindowName = MsgSpy1.FormName

		If (MessageName = "WM_MOUSEMOVE") Then
			pnl3dMessage = "Mouse is over " & WindowName
			pnl3dXPos = Str$(MsgSpy1.LOWORD)
			pnl3dYPos = Str$(MsgSpy1.HIWORD)
		End If
	End If

End Sub
```