Contents

Overview

What Is VB Messenger?

VB Messenger is a custom control to be used in Visual Basic applications. VB Messenger allows a programmer to tap into the power of Windows by intercepting Windows messages while still providing the ease of use of Visual Basic.

VB Messenger allows the Visual Basic programmer to *subclass* a Visual Basic form or control (or any Windows control or window) to intercept messages that are intended for the form or control.

What Is Subclassing?

Subclassing is a technique used in Windows programming to programatically add additional functionality to an existing window or control.

The way it works is you tell Windows that you want your routine to be the first to get messages for a specific window. Then, in your routine you can code specifically for messages that you wish to process. Then you either pass the message on to the actual window you are subclassing or "throw it away." (The latter will assume that you did everything that was required to assure the integrity of the window.)

Basically, that is exactly what VB Messenger does. You specify which form or control to subclass and VB Messenger takes care of all the complexities of performing the actual subclass. Instead of having Windows call a routine in your program to process the message, VB Messenger instructs Windows to call its own message routine. VB Messenger then passes the information to you via a Visual Basic custom *event*.

Why can't you just instruct Windows to call one of *your* Visual Basic routines? Actually, this is impossible to do with Visual Basic code. In order for Windows to call your routine, it must have the actual address of the routine. Visual Basic does not provide the means for obtaining the address of a Visual Basic routine. Since VB Messenger is written in C, it can pass the address of its internal function to Windows, since C provides the means for getting a function address.

What are Windows Messages?

Windows messages are the essence of how the Microsoft Windows Operating Environment works. Each form, window or control that you see on the screen can be thought of as an independent extension of Windows. Windows knows of the existence of each window or control. Before the programmer instructs Windows to create the window, the programmer registers a function within the programmer's code that is associated with the window. Windows can then control the behavior and characteristics of the window when necessary.

For instance, when a window gets overlapped by another window and then comes back into the foreground, it is not normally the programmer's responsibility to redraw certain window elements such as the caption, min/max buttons, etc. The Windows environment actually does all of this. However, if the programmer has drawn a graph in the client area of the window, it is now the responsibility of the programmer to redraw the graph when it has been overwritten. Windows cannot automatically save all the graphics or text associated within the client area of a window.

So, how does the programmer know when this has to occur? Well, that is when Windows messages comes into play. Each time an event occurs in Windows, such as the repainting of an overlapped window, the Windows program is notified. This is done via a Windows message. A Windows message is nothing more that a predefined constant number (defined by Windows, see the Microsoft Windows 3.1 SDK). Since, as stated before, the programmer told Windows of a function to call when it needs to, Windows can alert the program by calling this function and passing a message to it. Windows defines many messages and some can even be defined by the programmer. Common messages are WM_PAINT, WM_SIZE, WM_CREATE, etc. The WM_PAINT message is the one that is sent to the programmer when the event as stated above occurs. The programmer tests to see if the message equals WM_PAINT, if so the programmer then calls a function to redraw the graph.

Simple? Although there are many programmers out there that can write Windows programs using C or C++ with their hands tied behind their backs, Windows programming is still quite complex. However, with the advent of Visual Basic, more and more people are starting to develop Windows applications more easily and more productively. Visual Basic hides the programmer from many complexities including intercepting Windows messages.

Visual Basic treats Windows messages as *events*. If you have programmed in Visual Basic, you have undoubtedly had experience using events. These events are actually Visual Basic's way of presenting you with Windows messages. The only problem is that Visual Basic only provides a predefined set of events (or messages) to intercept. And furthermore, Visual Basic does not provide a way to return a value to Windows after the event has been triggered.

How does one overcome this limitation? We all know that with simplicity comes limitations. Visual Basic does not let you create your own custom events unless you write a custom control in C to do so. VB Messenger is the control that will help you. It allows you to provide functionality in your Visual Basic application that you could not normally do without programming in C. With it you can specify what message you want to trap (there are literally hundreds to choose from) and VB Messenger will trigger your custom event when the message occurs.

Why Would I Use VB Messenger?

VB Messenger extends Visual Basic allowing you to define which messages you would like to intercept for a given form or control. VB Messenger can trap a message for any window that is loaded in memory including those that are not in your Visual Basic application.

VB Messenger can be used for a countless number of reasons. One practical use may be to display a Dynamic Information Line (DIL) on a status bar. A DIL is a line that reports context sensitive help when the mouse moves over a portion of the screen. For example, as seen in many Windows applications, while the user is selecting a menu item, your application can detect this and display a message on your status bar.

VB Messenger can also be used as a diagnostic tool. For instance, if you wanted to create a program that prints the literal form of a message (i.e., "WM_PAINT") for a specific window to see what it is doing, you can use VB Messenger to trap that window and report back each message. You can then translate the message to a string by using VB Messenger's built in message decoder, which translates the message number to a literal.

Or, a more complicated use could be to synchronize two or more list boxes. For instance, if you have a requirement to display column headings for a list of columnar data in a list box and allow horizontal scrolling, you cannot do this with a standard list box because when the list box scrolls horizontally, the column headings are no longer above the correct columns. However, you could stack two list boxes vertically, one on top for column headings and one on the bottom for the data. Using VB Messenger, you can detect when one list box scrolls and then programatically scroll the other one the same amount and direction.

Your VB Messenger disk comes with several samples of Visual Basic code that perform a variety of different functions using the VB Messenger Custom Control.

Do I Need to Know Windows Programming?

To answer this question, first you must realize that if you program in Visual Basic, you are a fully qualified Windows programmer. More and more, as with each new release of Visual Basic, Windows programming is becoming easier and easier. You no longer have to program in C or C++ to take full advantage of the Windows environment.

Visual Basic does shield you from much of the complexities of programming in the Windows environment. But nobody ever said that a Visual Basic programmer cannot access the same functionality in Windows as a C or C++ programmer can. You can go beyond the scope of Visual Basic and access the Windows APIs directly. And now since you have VB Messenger, there is nothing in your way of writing a robust, killer application using Visual Basic.

To find out about which Windows messages you can trap and what they all mean, browse through the on-line Windows SDK help file that comes with Visual Basic. It will tell you exactly which messages are out there. With a little exploring (and alot of courage) you can discover new ways to increase the functionality of your applications and have them behave like any of the best Windows programs you can find.

One word of caution is necessary. Accessing the Windows API directly is always a dangerous process if you are not careful. Since Windows is not a protected operating environment, you can easily hang your system if you pass the wrong parameters with a message or as a return value from a message. Visual Basic shields you from most of these possibilities the best it can. But if you go around this shield, do it carefully. And remember, SAVE YOUR WORK OFTEN!

Using VB Messenger

Adding VB Messenger to Your Application

All Visual Basic custom controls are loaded into your project from within the Visual Basic environment. Once a custom control is added to your project and your project is saved, it will always load whenever you load that project.

To add the VB Messenger custom control to your project, choose '**Add File**' from the '**File**' menu in Visual Basic. Then, enter the name of the design time VB Messenger custom control, and its fully qualified path if necessary. The VB Messenger icon should now appear in your Visual Basic Toolbox.

To use VB Messenger, you must first add it to a form in your project. When you add the VB Messenger custom control to your form, it will appear on your form as an icon similar to the one that appears in the Toolbox. Although you can reposition the icon anywhere you like on the form, the position of the icon is not important. When you run your application, the custom control becomes *invisible* and therefore cannot be seen by the user.

Refer to the *'Loading Custom Controls'* section in the *Visual Basic Programmer's Guide* for a detailed explanation of adding controls to your project.

How Do I Use VB Messenger? - A Guided Tour

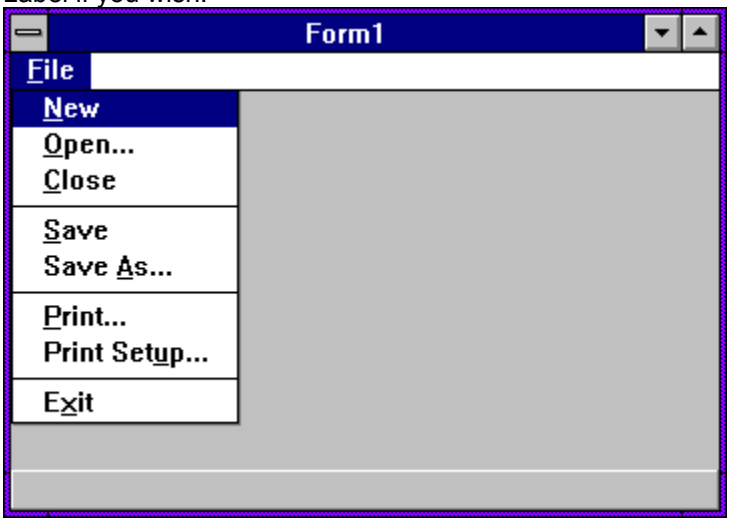
VB Messenger is used to subclass a form or control in Visual Basic to intercept the Windows messages that are associated with the form or control.

Once added to your form, VB Messenger is ready to use. Most of VB Messenger's properties can be set via the property window in design mode of Visual Basic.

The following instructions provide you with a sample walk through on using VB Messenger with your application. It walks you through the creation of a sample program that illustrates the fundamental usage of the VB Messenger Custom Control. The program uses VB Messenger to trap the **WM_MENUSELECT** message that is sent to the main form when the user highlights a menu item in a drop down menu. Then the program will display information about the menu item selected on a message line at the bottom of the screen in a status bar.

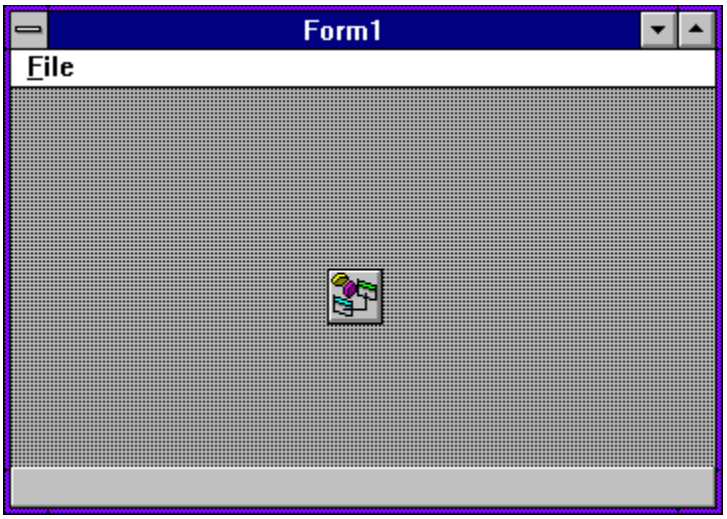
Step 1.

First create the following form with the following drop down menu. The bottom of the form contains a 3D Panel control with the Align property set to Align Bottom. You may substitute a standard Visual Basic Label if you wish.



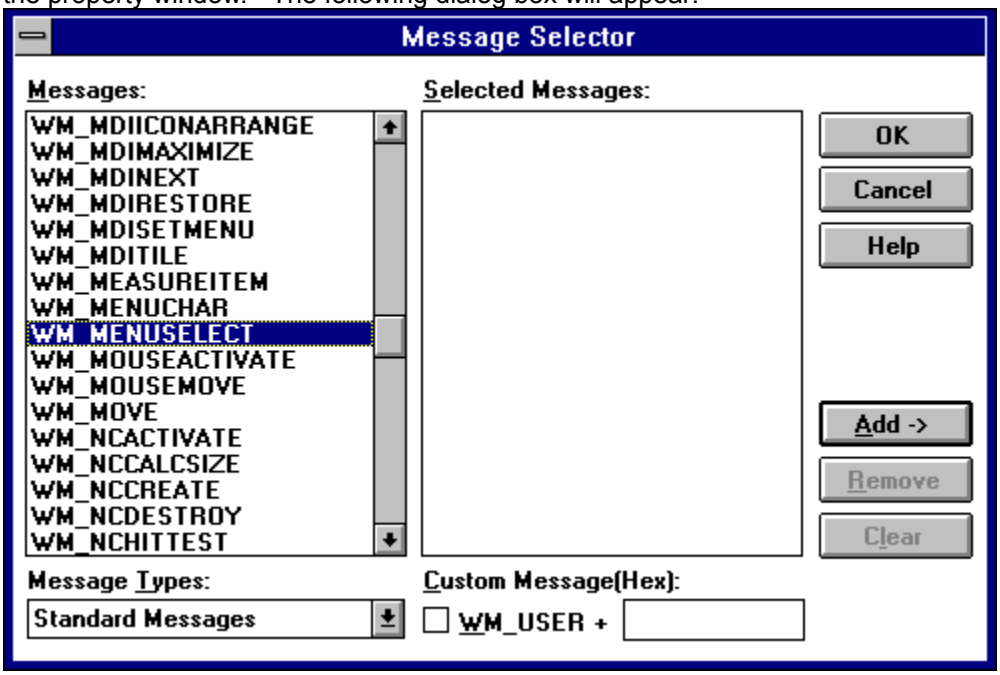
Step 2.

Add the VB Messenger Custom Control to the new form.



Step 3.

Bring up the property window for the new VB Messenger Custom Control. Select the **(Message Selector)** property from the property window. Click once on the ellipses ('...') button next to the text in the property window. The following dialog box will appear:



Scroll through the **Messages** list box and search for **WM_MENUSELECT**. Select this message by double clicking on it with the mouse or by pressing the **Add->** button. The message will then appear in the **Selected Messages** list box. Click on **OK** to save the selection and close the Message Selector dialog box.

Step 4.

Add the following code to the Form1's Form_Load procedure:

```
VBMsg1.SubClasshWnd = Form1.hWnd
```

When the form is loaded, this code will be executed instructing VB Messenger to subclass the main form.

VB Messenger immediately starts intercepting messages at this point. To turn off the message processing, just set the SubclasshWnd property to zero.

Step 5.

Now add the following code to the VBMsg1_WindowMessage procedure:

```
Panel3D1.Caption = "wParam =" & Str$(wParam)
```

wParam is passed to this procedure by VB Messenger.

Step 6.

Run the program. Use the mouse to select through the different menu items and watch how VB Messenger intercepts the message and allows you to display the information about the menu item on the status bar.

This example detects when the user is selecting a menu item and displays the wParam parameter associated with the message (WM_MENUSELECT) at the bottom of the screen. You can use this number in your program to reference a line of text that may be used to describe the particular function on the menu. Then you can display this text on the bottom of the screen, very similarly to other Windows programs.

See the sample project **MENU.MAK**.

VB Messenger Custom Control Reference

Description	The VB Messenger Custom Control allows you to subclass a form or control to receive and/or intercept its messages. 
File Name	VBMSG.VBX
Object Type	VBMsg

Related Topics:

Properties, Events, and Methods
Properties Reference
Events Reference

Properties, Events, and Methods

All of the properties, events, and methods for VB Messenger are listed in the tables below. All standard Visual Basic properties, events, and methods are denoted with an asterisk(*) and can be found documented in the *Visual Basic Language Reference* that comes with Visual Basic.

Properties

About	AddMessage	ClearMessages
*Height	HiWord	*hWnd
*Index	*Left	LoWord
lParam	lParam2String	MessageCount
MessageList	MessageSelector	MessageText
MessageTypes	*Name	*Parent
PostDefault	PostMessage	RemoveMessage
ReturnVal	SendMessage	String
SubclasshWnd	*Tag	*Top
*Width	wParam	

Events

WindowMessage	WindowDestroyed

Methods

VB Messenger does not support any methods.

Properties Reference

The following is a detailed reference of all the properties supported by VB Messenger.

Related Topics:

About Property

Description Displays version information about the VB Messenger Custom Control.

Usage Double click on the ellipses ('...') button next to the property text to activate the about dialog box.

Remarks Available only at design time.

Data Type N/A

AddMessage Property

Description Adds a message to the MessageList property.

Usage [*form.*]*VBMsg.***AddMessage**[= *message&*]

Remarks To specify which messages are intercepted by VB Messenger at runtime, you must use this property. Each time you set the AddMessage property to a message, a new message gets appended to the end of the MessageList property array.

Example:

```
VBMsg.AddMessage = WM_PAINT
VBMsg.AddMessage = WM_SIZE
VBMsg.AddMessage = WM_CLOSE
```

Available only at runtime and is write only.

Data Type Long

See Also MessageSelector, RemoveMessage, ClearMessages

ClearMessages Property

Description Clears all messages from the MessageList property.

Usage [*form.*]*VBMsg.***ClearMessages** = *True*

Remarks Setting this property to True (or any integer value) clears all the messages from the MessageList property. This is the equivalent of using RemoveItem for all messages in the MessageList.

Only available at runtime and is write only.

Data Type Boolean

See Also RemoveMessage, AddMessage, MessageSelector

HiWord Property

Description Returns or sets the high-order word of the 32-bit long integer value in the lParam property.

Usage [*form.*]*VBMsg.***HiWord**[= *value%*]

Remarks

This property is used primarily in conjunction with the LoWord property to create the lParam property. This property is useful when you need to send the lParam parameter with a message to a window that calls for the high-order word to be a certain value.

Setting this value to an integer causes VB Messenger to combine this value with the LoWord property and set the lParam property to the result. VB Messenger performs a concatenation of the two 2-byte integer values to produce the 4-byte integer lParam property.

Setting the lParam property to a long integer causes VB Messenger to parse out two 2-byte values and place the results in LoWord and HiWord respectively.

Example:

```
VBMsg1.LoWord = Form1.hWnd
VBMsg1.HiWord = 100
Print "The resulting lParam is:"; VBMsg1.lParam
```

Available only at runtime.

Data Type Integer

See Also LoWord, lParam, SendMessage, PostMessage

LoWord Property

Description Returns or sets the low-order word of the 32-bit long integer value in the lParam property.

Usage [*form.*]*VBMsg.***LoWord**[= *value%*]

Remarks

This property is used primarily in conjunction with the HiWord property to create the lParam property. This property is useful when you need to send the lParam parameter with a message to a window that calls for the low-order word to be a certain value.

Setting this value to an integer causes VB Messenger to combine this value with the HiWord property and set the lParam property to the result. VB Messenger performs a concatenation of the two 2-byte integer values to produce the 4-byte integer lParam property.

Setting the lParam property to a long integer causes VB Messenger to parse out two 2-byte values and place the results in LoWord and HiWord respectively.

Example:

```
VBMsg1.LoWord = Form1.hWnd
VBMsg1.HiWord = 100
Print "The resulting lParam is:"; VBMsg1.lParam
```

Available only at runtime.

Data Type Integer

See Also HiWord, lParam, SendMessage, PostMessage

lParam Property

Description This property represents the 32-bit long value of the Windows message structure.

Usage [*form.*]*VBMsg.***lParam**[= *value&*]

Remarks This property is used primarily in conjunction with the SendMessage and PostMessage properties as a parameter for sending messages directly to a subclassed window.

This property can also be used to parse out the low-order and high-order word values of any 32-bit long integer. The results can be found in the HiWord and LoWord properties respectively.

Example:

```
' select a range of items in a multi-select list box
VBMsg1.wParam = True
VBMsg1.LoWord = 0
VBMsg1.HiWord = List1.ListCount

'The concatenated value is now in lParam property
VBMsg1.SendMessage = LB_SELITEMRANGE
```

Available only at runtime.

Data Type Integer

See Also wParam, SendMessage, PostMessage

lParam2String Property

Description This property converts a 32-bit address to a Visual Basic string.

Usage [*form.*]*VBMsg.***lParam2String**[= *value&*]

Remarks Setting this value to a valid 32-bit far segment address (stored in a long integer) causes VB Messenger to place the data pointed to by the address into the String property. The length of the resulting String property is determined by the first occurrence of an ASCII 0 in the data.

WARNING! Use this property very carefully. Setting this to an invalid pointer could result in undesirable results such as a GPF or loss of data. Do not set this property to anything else except a valid pointer.

Generally this property is used to convert the lParam message parameter passed from within the WindowMessage event procedure to a Visual Basic string.

Available only at runtime and is write only.

Example:

```
'... from within the VBMsg1_WindowMessage proc ...
          .
          .
VBMsg1.lParam2String = lParam
Print "The resulting string is: " & VBMsg1.String
```

Data Type Long

See Also lParam

MessageCount Property

Description Returns the number of messages in the MessageList property array.

Usage [*form.*]*VBMsg.***MessageCount**

Remarks Available only at runtime and is read only.

Data Type Integer

See Also MessageList

MessageList Property

Description Contains a list of all messages to be intercepted by VB Messenger.

Usage [*form.*]*VBMsg.***MessageList(***index)*[= *message&*]

Remarks This property array contains all the messages set by either AddMessage or at design time by the MessageSelector dialog. Each time a new message is added, the message gets appended to the end of this list, increasing the count by one. To access any message in the list, you must specify the index of the array.

You can also change messages in the list by assigning the specific element in the property array.

Example:

```
' changes the 3rd message (0 based) in the list
' to WM_CLOSE
VBMsg.MessageList(2) = WM_CLOSE
```

Available only at runtime. The first element in the array is at index 0.

Data Type Long

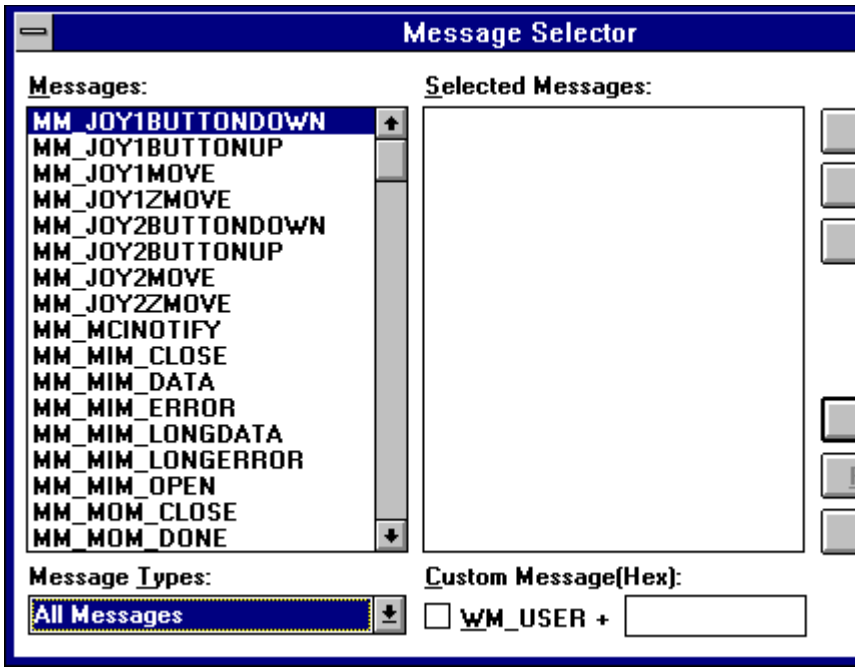
See Also MessageCount, MessageSelector, AddMessage, RemoveMessage, ClearMessages

MessageSelector Property

Description Displays a dialog box from which you can manage the list of messages to be intercepted by VB Messenger.

Usage Double click on the ellipses ('...') button next to the property text to activate the about dialog box.

Remarks Clicking on the ellipses in the property window display the Message Selector dialog box. The Message Selector allows you to add and remove standard and custom messages to the MessageList Property array at design time.



The **Messages** list box contains all available standard messages. The **Selected Messages** list box on the right contains your selected messages to be intercepted. Clicking on the **Add**, **Remove** or **Clear** button allows you to manage the selected messages.
You can filter the types of standard messages to be displayed in the **Messages** list box by selecting from the **Message Types** drop down list.
You can specify a custom message that does not appear in the **Messages** list of standard messages by entering the message value (in hex) into the **Custom Message** edit box. Optionally, you can check the **WM_USER +** check box to add the value of WM_USER to the entered custom message.
Available only at design time.

Data Type N/A

See Also MessageList, MessageCount, AddMessage, RemoveMessage, ClearMessages

MessageText Property

Description Converts a message value to the corresponding text (i.e., "WM_PAINT") as defined by the Windows 3.1 SDK.

Usage [*form.*]*VBMsg.***MessageText(***message&)*

Remarks This property array contains all the messages in literal form. You can specify the message description to retrieve by indicating the message value as the index to the property array.

Example:

```
Const WM_CLOSE = &H10

X$ = VBMsg.MessageList(WM_CLOSE)
' X$ now equals "WM_CLOSE"
```

Available only at runtime and is read only.

Data Type String

MessageTypes Property

Description Allows you to specify how VB Messenger interprets the MessageList property.

Usage [*form.*]*VBMsg.***MessageTypes**[= *setting%*]

Remarks Use the MessageTypes property to instruct VB Messenger when to fire an event in accordance to when the messages in the MessageList property array are detected.

The MessageTypes settings are as follows:

Setting	Description
0	Intercept selected messages in the MessageList property only.
1	Intercept all messages (ignore MessageList property).
2	Do not intercept any messages.
3	Intercept all messages except those selected in the MessageList property array.

Data Type Integer (Enumerated)

PostMessage Property

Description Posts a message to the Windows message queue for the subclassed window.

Usage [*form.*]*VBMsg.***PostMessage**[= *message&*]

Remarks Setting this property will cause VB Messenger to post the specified message for the subclassed window to the Windows message queue. VB Messenger uses the properties wParam and lParam as the 16-bit word and 32-bit long parameters for the message. The return value of the posted message can be obtained in the ReturnVal property.

If the SubclasshWnd property is not set, no message will be posted.

Example:

```
Const WM_CLOSE = &H10

VBMsg1.wParam = 0
VBMsg1.lParam = 0
VBMsg1.PostMessage = WM_CLOSE
```

Available only at runtime and is write only.

Data Type Long

PostDefault Property

Description Determines whether VB Messenger should send the messages to your application before or after Windows default processing.

Usage [*form.*]*VBMsg.***PostDefault**[= {**True**|**False**}]

Remarks Setting this property to **True** causes VB Messenger to call the Windows default processing for all messages *before* the message is sent to your application. Setting this to **False** will cause VB Messenger to call the Windows default processing for all messages *after* the message is sent to your application.

If you plan on returning a value to Windows after processing a message in you WindowMessage event procedure, you must set this property to True or the return value will be ignored.

Data Type Boolean (Integer)

See Also WindowMessage Event

RemoveMessage Property

Description Removes a message from the MessageList property.

Usage [*form.*]*VBMsg.***RemoveMessage**[= *index%*]

Remarks Setting this property to the message value of a message in the MessageList property array will remove the message from the list.

Example:

```
' Remove message from the MessageList property
VBMsg1.RemoveMessage = WM_CLOSE
```

Available only at runtime and is write only.

Data Type Long

See Also ClearMessages, AddMessage

SendMessage Property

Description Sends a message directly to the subclassed window bypassing the message queue.

Usage [*form.*]*VBMsg.***PostMessage**[= *message&*]

Remarks Setting this property will cause VB Messenger to send the specified message directly to the subclassed window. VB Messenger uses the properties wParam and lParam as the 16-bit word and 32-bit long parameters for the message. The return value of the message can be obtained in the ReturnVal property.

If the SubclasshWnd property is not set, no message will be sent.

Example:

```
' select a range of items in a multi-select list box
VBMsg1.wParam = True
VBMsg1.LoWord = 0
VBMsg1.HiWord = List1.ListCount
'The concatenated value is now in lParam property

VBMsg1.SendMessage = LB_SELITEMRANGE
```

Available only at runtime and is write only.

Data Type Long

ReturnValue Property

Description This property is set with the return value of the SendMessage or PostMessage property.

Usage [*form.*]*VBMsg.***ReturnVal**

Remarks

Example:

```
' This example sends an EM_GETLINECOUNT message to
' retrieve the number of lines in a multiline edit
' control and then sends an EM_LINESCROLL message
' to scroll the edit control so that the last line
' is displayed at the top of the edit control.

VBMsg1.SendMessage = EM_GETLINECOUNT

' Number of lines returned can be found in the
' ReturnVal property.
VBMsg1.wParam = 0
VBMsg1.LoWord = VBMsg1.ReturnVal - 1
VBMsg1.HiWord = 0
VBMsg1.SendMessage = EM_LINESCROLL
```

Available only at runtime and is read only.

Data Type Long

See Also SendMessage, PostMessage

String Property

Description This property is set with the resulting string after setting the lParam2String property. Setting this property returns an address which can be found in the lParam property.

Usage [*form.*]*VBMsg.***String**

Remarks If you set this property to a string, VB Messenger will return an address of the string in the lParam property. The address of the string will be valid for as long as VB Messenger is active or the string is replaced with a new string.

Example:

```
'... from within the VBMsg1_WindowMessage proc ...
          .
          .
VBMsg1.lParam2String = lParam
Print "The resulting string is: " & VBMsg1.String
```

Available only at runtime.

Data Type String

See Also lParam2String, lParam

SubclasshWnd Property

Description Set this property to the window handle (hWnd) of the form or control to subclass.

Usage [*form.*]*VBMsg.***SubclasshWnd** = [*handle%*]

Remarks Setting this property to a valid window handle immediately activates VB Messenger. All messages sent to the window associated with the handle from that point onward will be filtered by VB Messenger and the event WindowMessage will be fired for each.

Setting this property to zero will automatically disable the subclassing. Upon the destruction of the window (WM_DESTROY), this property is cleared and subclassing will terminate.

Available only at runtime.

Data Type Integer

wParam Property

Description This property represents the 16-bit integer value of the Windows message structure.

Usage [*form.*]*VBMsg.***wParam**[= *value%*]

Remarks This property is used primarily in conjunction with the SendMessage and PostMessage properties as a parameter for sending messages directly to a subclassed window.

Available only at runtime.

Data Type Integer

See Also lParam, SendMessage, PostMessage

Events Reference

The following is a detailed reference of all the events supported by VB Messenger.

Related Topics:

WindowMessage Event
WindowDestroyed Event

WindowMessage Event

Description VB Messenger fires this event each time one of the selected messages is detected for the subclassed window.

Syntax **Sub VBMsg_WindowMessage** (*hWindow* **As Integer**, *Msg* **As Integer**, *wParam* **As Integer**, *lParam* **As Long**, *RetVal* **As Long**, *CallDefProc* **As Integer**)

Remarks When a message that you wish to intercept is detected for the subclassed form, VB Messenger fires this event passing the message's parameters to the event procedure.

Parameter	Description
hWindow	Identifies the subclassed window.
Msg	The message that was detected.
wParam	The 16-bit word value associated with the message.
lParam	The 32-bit long value associated with the message.
RetVal	After you process the message, use this parameter if you wish to return a value to VB Messenger and bypass the default windows procedure. VB Messenger will then use this value as the return value.
CallDefProc	If this value is True, VB Messenger will call the default windows procedure for the subclassed control. If it is False, VB Messenger will not call the default procedure and return the specified return value in RetVal.

See Also WindowDestroyed Event

WindowDestroyed Event

Description VB Messenger fires this event unconditionally if the subclassed window is sent a WM_DESTROY message.

Syntax **Sub VBMsg_WindowDestroyed** (*hWindow* **As Integer**)

Remarks This event is fired when the subclassed window is destroyed. This is useful for code to clean up memory that you may have associated with the subclassed window. The window is automatically unhooked after this event is fired.

Parameter	Description
hWindow	Identifies the subclassed window.

See Also WindowDestroyed Event

VB Messenger API

Overview

VB Messenger comes with a set of API functions that you will need to process certain messages.

Several Windows messages require the programmer to be able to access data via pointers. Although in languages like C it is possible to provide pointers, it is not possible using Visual Basic. These API functions allow you to access data while "faking" pointers. VB Messenger uses long integers to represent the pointers. Since pointers are actually just 32-bit numbers (i.e., long integers), you can actually "fake" Windows by sending certain Windows API functions the long integer equivalent of a pointer as supplied by the VB Messenger API.

CAUTION: These routines require the use of pointers. Take care when using such routines as they may cause unpredictable results if used improperly. Do not pass invalid addresses to these routines. Doing so may cause a GPF or potential loss of data.

Special Note: You may notice that the Lib in the Declare statements below refer VBMSG.VBX. Why is this not referencing a DLL? Since a custom control (VBX) is really a DLL (with special routines in it so that Visual Basic can access it), functions can be called externally from them. So rather than supplying a separate DLL that you would need to include with your distribution, VB Messenger comes with a full set of functions built right into itself. All of the following functions can be called directly from the file VBMSG.VBX.

As with all DLLs and VBXs, the executable file must be either in the path, the current directory, or the Windows SYSTEM directory in order for Visual Basic to find and load them. See the Chapter 22, "*Calling Procedures in DLLs*" in the *Microsoft Visual Basic Programmer's Guide* for a further description on calling external procedures.

API Reference

The following section details the API functions available within VB Messenger.

Related Topics:

ptConvertUShort
ptCopyTypeToAddress
ptGetControlModel
ptGetControlName
ptGetIntegerAddress, ptGetLongAddress, ptGetStringAddress
ptGetIntegerFromAddress
ptGetLongFromAddress
ptGetStringFromAddress
ptGetTypeFromAddress
ptGetVariableAddress
ptHiWord
ptLoWord
ptMakelParam
ptMakeUShort
ptMessageToText
ptSetControlModel

ptConvertUShort

Description This function converts an unsigned integer value returned from a DLL to an long integer.

Declaration Declare Function **ptConvertUShort** Lib "VBMSG.VBX" (ByVal ushortVal As Integer) As Long

Remarks Since Visual Basic cannot represent an unsigned value, the value returned from the DLL may be negative. This function will convert it to a positive long integer value. If the number is already positive, its value will be copied directly.

Parameter	Description
ushortVal	The integer value to convert.

Return Value The function returns a long integer value representing the unsigned integer.

ptCopyTypeToAddress

Description This function copies the contents of a type variable to an area of memory.

Declaration Declare Sub **ptCopyTypeToAddress** Lib "VBMSG.VBX" (ByVal lAddress As Long, lpType As Any, ByVal cbBytes As Integer)

Remarks Since Visual Basic does not allow you to access memory directly, there is no way to change the contents of a structure passed as a pointer in a Windows API. You can use this function and ptGetTypeFromAddress to write the contents of a Type to an area of memory.

Parameter	Description
lAddress	A long integer representing the far address of the area of memory to write to.
lpType	The Type variable to write to memory.
cbBytes	The length of the Type variable. This can be determined using the Visual Basic Len function, i.e., cbBytes = Len(lpType).

CAUTION: This routine requires the use of a pointer. Do not pass an invalid address to this routine. Doing so may cause a GPF or potential loss of data.

ptGetControlModel

Description Fills the MODEL control model structure for a control.

Declaration Declare Function **ptGetControlModel** Lib "VBMSG.VBX" (ctl As Control, lpmodel As MODEL) As Long

Remarks By examining fields of this structure you can determine how a controls flags are set, and you also have access to its property and event information tables. You can also set values in this structure and change the behavior or style of a control.

Parameter	Description
ctl	The name of the control.
lpmodel	The MODEL type (structure) to fill.

Return Value The function returns a long integer value representing the far address of the control model structure.

ptGetControlName

Description Returns a string containing the name of a specified control.

Declaration Declare Function **ptGetControlName** Lib "VBMSG.VBX" (ctl As Control) As String

Remarks Visual Basic does not allow you to access the name of a control during runtime. This function allows you to get the name of a control dynamically at runtime.

Parameter	Description
ctl	The name of the control.

Return Value The function returns a string representing the name of the control.

ptGetIntegerAddress, ptGetLongAddress, ptGetStringAddress

Description Returns the address of a Visual Basic variable with stricter type checking.

Declarations Declare Function **ptGetIntegerAddress** Lib "VBMSG.VBX" Alias "ptGetVariableAddress" (ByVal var As Integer) As Long

Declare Function **ptGetLongAddress** Lib "VBMSG.VBX" Alias "ptGetVariableAddress" (ByVal var As Long) As Long

Declare Function **ptGetStringAddress** Lib "VBMSG.VBX" Alias "ptGetVariableAddress" (ByVal var As String) As Long

Remarks Each of these functions return a 32 bit address of a variable. These function declarations are provided to give better parameter checking when using this function.

Parameter	Description
var	The variable to get the address of. Can either be an integer, long, or string depending upon the declaration used. For Types, use the ptGetVariableAddress API.

Return Value These functions returns a 32-bit address of the variable.

ptGetIntegerFromAddress

Description Returns the 16-bit integer value from the data at the address specified.

Declaration Declare Function **ptGetIntegerFromAddress** Lib "VBMSG.VBX" (ByVal address As Long) As Integer

Remarks

Parameter	Description
address	The 32-bit far address of the integer.

Return Value Returns the integer value associated with the address.

ptGetLongFromAddress

Description Returns the 32-bit long integer value from the data at the address specified.

Declaration Declare Function **ptGetLongFromAddress** Lib "VBMSG.VBX" (ByVal address As Long) As Long

Remarks

Parameter	Description
address	The 32-bit far address of the long integer.

Return Value Returns the long integer value associated with the address.

ptGetStringFromAddress

Description Returns a string from the data at the address specified.

Declaration Declare Function **ptGetStringFromAddress** Lib "VBMSG.VBX" (ByVal address As Long) As String

Remarks The string located at the address specified must end with a terminating zero.

Parameter	Description
address	The 32-bit far address of the string.

Return Value Returns the string associated with the address.

ptGetTypeFromAddress

Description Returns a Type structure from the data at the address specified.

Declaration Declare Sub **ptGetTypeFromAddress** Lib "VBMSG.VBX" (ByVal address As Long, typevar As Any, cbBytes As Integer)

Remarks The data located at the address specified will be copied into the Type structure variable defined by the calling program. Only the number of bytes specified will be copied. Do not specify more bytes then are actually allocated. Doing so may produce unpredictable results such as a GPF or loss of data.

Parameter	Description
address	The 32-bit far address of the data.
typevar	The user defined Type variable to copy the data into.
cbBytes	The number of bytes to copy.

ptGetVariableAddress

Description Returns the address of a Visual Basic variable.

Declaration Declare Function **ptGetVariableAddress** Lib "VBMSG.VBX" (variable As Any) As Long

Remarks This function returns a 32 bit address of any variable or Type. Any type of variable can be used.

Parameter	Description
variable	The variable or Type to get the address of.

Return Value This function returns a 32-bit address of the variable or Type.

ptHiWord

Description This function parses out the high-order 16-bit word value of a 32-bit long integer.

Declaration Declare Function **ptHiWord** Lib "VBMSG.VBX" (ByVal lParam As Long) As Integer

Remarks This API provides the same functionality as the property HiWord.

Parameter	Description
lParam	The 32-bit long integer value to parse.

Return Value Returns a 16-bit integer representing the high-order of the 32-bit long value.

ptLoWord

Description This function parses out the low-order 16-bit word value of a 32-bit long integer.

Declaration Declare Function **ptLoWord** Lib "VBMSG.VBX" (ByVal lParam As Long) As Integer

Remarks This API provides the same functionality as the property LoWord.

Parameter	Description
lParam	The 32-bit long integer value to parse.

Return Value Returns a 16-bit integer representing the low-order of the 32-bit long value.

ptMakeIParam

Description This functions creates an unsigned long integer for use as an lParam parameter in a message by concatenating two integer values, specified by the wLow and wHigh parameters.

Declaration Declare Function **ptMakeIParam** Lib "VBMSG.VBX" (ByVal wLow As Integer, wHigh As Integer) As Long

Remarks

Parameter	Description
wLow	Specifies the low-order word of the new long value.
wHigh	Specifies the high-order word of the new long value. addressThe 32-bit far address of the data.

Return Value The return value specifies a long-integer value.

ptMakeUShort

Description This function converts a signed long integer value to an unsigned integer value.

Declaration Declare Function **ptMakeUShort** Lib "VBMSG.VBX" (ByVal longVal As

Long) As Integer

Remarks The value that is returned can be sent to a Windows API function that requires a USHORT or an unsigned int parameter. Upon inspection of the return value you may see that it is negative. Visual Basic does not have a means to represent an unsigned value, it therefore appears negative. When passed to function in a DLL, the number will be converted to unsigned.

Parameter	Description
longVal	The long integer value to convert.

Return Value The function returns an integer value.

ptMessageToText

Description Returns the literal description of a message number as define by the Windows SDK.

Declaration Declare Function **ptMessageToText** Lib "VBMSG.VBX" (ByVal message As Long) As String

Remarks This routine translates the message number to the literal text description of the message. This function is useful in developing a diagnostic program that detects all messages for a specific window and displays the message as text (i.e., WM_PAINT instead of &H10).

Parameter	Description
message	The message number.

Return Value The message string.

ptSetControlModel

Description Writes the contents of a modified MODEL control model structure to a control.

Declaration Declare Sub **ptSetControlModel** Lib "VBMSG.VBX" (ctl As Control, lpmodel As MODEL)

Remarks By examining fields of this structure you can determine how a controls flags are set, and you also have access to its property and event information tables. You can also set values in this structure and change the behavior or style of a control.

Parameter	Description
ctl	The name of the control.
lpmodel	The MODEL type (structure) to write to the control.