Message.VBX & MsgMore.DLL Help

Overview of Message.VBX & MsgMore.DLL

Message.VBX and MsgMore.DLL allow you to easily intercept ANY window messages; VB only responds to a few messages (called Events) per control. With this added capabilty you can respond to right-mouse clicks and double-clicks, set minimum and maximum window re-size dimensions, respond to custom system menu items, display statusbar messages based on menu selection (before clicking), move captionless forms, etc. Add a powerful new control dimension to your programming projects with "Message". You can actually subclass controls directly from within VB. And, Message.VBX is invisible at run-time (like the VB timer control).

The VB demonstration project and the online HLP file explain how to use "Message" so that you can quickly incorporate this new, powerful feature into your own projects. And, the installation package demonstrates usage of our VSWpro Deluxe (Visual Setup Wizard Professional) toolkit.

Copyright Notice

Message.VBX & MsgMore.DLL
Copyright © 1995 by Digital PowerTOOLS

This product is distributed under the shareware concept. Shareware Registration is mandatory for continued use of this product.

And, as shareware, you may evaluate this product for a period of no more than 30 days. After this time you must either register it or remove it from your system. Failure to comply with this condition is a violation of United States and international copyright law.

This program is fully functional; however, you will receive the latest version plus other benefits upon registering. Refer to the included SharWare.WRI file for a definition of shareware and a full description of our shareware policy.

Message.VBX Properties

Standard VB control properties (design and run-time) supported by Message.VBX are:

HelpContextID	Name
Index	Tag
Left	Top

New properties (run-time only) supported by Message.VBX are:

hWindow

hWindow tells Message.VBX what hWnd to intercept messages from. You should set this property in the Load event of the form that contains Message.VBX A typical usage might be: Message1.hWindow=me.hWnd.

Status

Message.VBX creates an array to handle 65536 distinct window messages. Each element in this arrary corresponds directly to a specific message. These message constants are declared in the Message.Bas file; you can add more as new versions of Windows become available that support additional messages. Although each Message.VBX can respond to ALL messages for a window or control, you'll typically just want to intercept a few of the messages in order to alter a control's functionality or to simply respond to messages that are not normally handled as VB Events by the form or control. Also, you should set this property in the Load event of the form that contains Message.VBX. A typical usage might be: Message1.Status(WM_MenuSelect)=True. Insert the message constant that you want to intercept in the parantheses and the value to True to enable interception.

"Message" Support Documents

Included with the Message.VBX and Message.DLL demo are five support files in WRI format; you can also access these files directly from the Help menu of the demonstation application. These files are:

DpCt0195.WRI (a description of other Digital PowerTOOLS shareware products)

EvalFrm.WRI (a shareware evaluation form for you to grade this product)

OnlineRg.WRI (explains how to register online through CompuServe)

OrderFrm.WRI (a convenient order form with bonus and upgrade discounts)

SharWare.WRI (a definition of shareware and a description of our policy on shareware)

Message.VBX Events

Message.VBX supports a single event, Receive. The values returned by Receive are:

Msg%	This is the message that has been intercepted. It's also going to be one of the messages you enabled with the Status property, e.g., WM_MenuSelect or WM_GetMinMaxInfo. You can use this value in Select Case or If ... then conditionals in order to screen multiple messages from a single instance of Message.VBX. Refer to the demo for more explicit usage application.
wParam%	This is the wParam integer returned by Windows. The meaning of wParam will vary depending on the message that you're intercepting.
lParam&	This is the lParam long value returned by Windows. The meaning of lParam will vary depending on the message that you're intercepting.
UseRetVal%	Set this integer to "1" to enable use of your own return value. Set it to "0" to use the default return value. If you do not set this value in code, it will be "0" -- use the default return value.
RetVal%	This is your own custom return value from the message.

A typical Receive event would be:

```
Sub Message1_Receive (Msg As Integer, wParam As Integer, lParam As Long, UseRetVal As Integer, RetVal As Long)

... your code here (refer to the demo examples on each form)

End Sub
```

Message.VBX Caution!!!

Message.VBX provides VB programmers with capabilities that are normally only available within the C programming environment. As such, you not only have tremendous control capability, but also a high risk of locking up the system. You should ONLY use Message.VBX if you are knowledgeable of window message processing. However, a few demos are included that you can copy directly into your projects. With an extensive knowledge of message processing, you can actually use Message.VBX to perform sophisticated control subclassing ... from within VB.

You should NOT use SendMessage from within a Receive event, use PostMessage instead. Don't access any control properties while in the Receive event as this can result in an internal SendMessage command.

You should make certain that values sent with MessageDataSet match exactly in size with the value received with MessageDataGet. Use as little code as possible within the Receive event because this a time-critical response.

Always save your project files BEFORE trying new Receive message processing, in case you've created an error situation that could potentially cause a GPF.

External Functions

MsgMore.DLL and Message.VBX should be in your Windows\System directory or in the Path. Many functions accept one or more command strings (command strings are case-insensitive and the function will return -255 or "Invalid Command" if the command is not recognized).

NOTE: Functions return a value directly and the parameters must be included in parantheses. Subs may return a value in a VB string and the parameters must NOT be included in parantheses.

NOTE: Some SUBS require you to set ReturnString$=space$(255) before calling and TrimAtNull(ReturnString$) after calling. This is because VB does not end characters in a NULL (chr$(0)) character. The DLL could write directly to a new VB string avoiding this situational requirement, however, this would prevent Messsage from being used by other Windows programming languages. As is, Message can be safely used from any Windows programming language including: C, C++, PowerPoint, dbFast, dBase for Windows, CA-Realizer, etc. (and, of course, Visual Basic).

Declare Sub MessageDataGet Lib "Message.VBX" (ByVal lParam&, ByVal DataSize%, DataType As Any)
used within a Receive event to obtain pointer data normally placed into a data structure

Declare Sub MessageDataSet Lib "Message.VBX" (ByVal lParam&, ByVal DataSize%, DataType As Any)
used within a Receive event to change pointer data normally placed into a data structure

Declare Function GetVersionInfo% Lib "MsgMore.DLL" (ByVal Application$, ByVal VersionType$)
determines version of assorted files
Application$="Windows", "Dos", or "Message"
VersionType$=major, minor, or full (full=major*100+minor)

Declare Function GetCPU% Lib "MsgMore.DLL" ()
returns 286, 386, or 486 as an integer determining the system CPU (actually, 486 is 486 or better)

Declare Function IsMathPresent% Lib "MsgMore.DLL" ()
returns -1 (TRUE) if a Math Coprocessor is present, returns 0 (FALSE) otherwise

Declare Function GetStateOfKey% Lib "MsgMore.DLL" (ByVal KeyName$)
returns -1 (TRUE) if the key is depressed; returns 0 (FALSE) otherwise
KeyName$="ScrollLock", "NumLock", "CapsLock", "Lshift", "Rshift", "Control", or "Alt"

Declare Function GetFreeMem& Lib "MsgMore.DLL" (ByVal InfoType$)
determines assorted memory availabilities
InfoType$="system", "gdi", "user", "space", or "contiguous"
NOTE: "system", "gdi", and "user" return a percentage free
("gdi" is graphics workspace, "user" is module space, and "system" is the lower of these two)
"space" and "contiguous" return the number of bytes (divide by 1024 to obtain kb free)
ret&=GetFreeMem("contiguous") also clears any unused, discarded memory blocks

Declare Function IsModeEnhanced% Lib "MsgMore.DLL" ()
returns -1 (TRUE) if in 386 Enhanced Mode, returns 0 (FALSE) if in Standard Mode

Declare Function GetHighByte% Lib "MsgMore.DLL" (ByVal OrginalInt%)

Declare Function GetLowByte% Lib "MsgMore.DLL" (ByVal OrginalInt%)

Declare Function GetHighWord& Lib "MsgMore.DLL" (ByVal OrginalLong&)

Declare Function GetLowWord& Lib "MsgMore.DLL" (ByVal OrginalLong&)

Declare Function TestBit% Lib "MsgMore.DLL" (ByVal TestInteger%, ByVal BitToTest%)
tests to see if the selected bit (0-15) is set
returns -1 (TRUE) if set; returns 0 (FALSE) if not

Declare Function GetColorValue% Lib "MsgMore.DLL" (ByVal GetColor$, ByVal RGBcolor&)
derives the specified color from an RGB color
GetColor$="blue", "red", or "green"

Declare Function WordColor& Lib "MsgMore.DLL" (ByVal ColorName$)
allows color selection with words instead of numbers

works anywhere that you would normally use QBCOLOR or RGB
Basic Color Words (include appropriate quotes around string):
Black,Blue,Green,Cyan,Red,Magenta,DarkYellow,LightGray,DarkGray,
BrightBlue,BrightGreen,BrightCyan,BrightRed,BrightMagenta,BrightYellow,BrightWhite
System Color Words (include appropriate quotes around string):
ActiveBorder,ActiveCaption,AppWorkSpace,BackGround,BtnFace,BtnHighlight,
BtnShadow,BtnText,CaptionText,GrayText,Highlight,HighlightText,InactiveBorder,
InactiveCaption,InactiveCaptionText,Menu,MenuText,Window,
WindowFrame,WindowText