Wildcat

Software

WinDLL Fastrack: Message Functions

For information on how to use help:
choose Help - Using Help.

Messages covered in this release.

Edit Controls

- EM_CANUNDO
- EM_EMPTYUNDOBUFFER
- EM_GETMODIFY
- EM_LIMITTEXT
- EM_UNDO

List Box Controls

- LB_FINDSTRING
- LB_GETCURSEL
- LB_GETSEL
- LB_GETTOPINDEX
- LB_INSERTSTRING
- LB_RESETCONTENT
- LB_SELECTSTRING
- LB_SETCURSEL
- LB_SETTOPIND

Combo Box Controls

- CB_FINDSTRING
- CB_GETCURSEL
- CB_INSERTSTRING
- CB_LIMITTEXT
- CB_RESETCONTENT
- CB_SELECTSTRING
- CB_SETCURSEL
- CB_SHOWDROPDOWN

General Windows Messages

- WM_TIMECHANGE
- WM_UNDO
- WM_WININICHANGE

Using the WIN_MSG Example Program

Currently demonstrated messages are for the Text, List Box and Combo Box controls. Text (Edit Messages) are prefixed with an 'EM_', List box Messages with a 'CB_', and Combo box Messages with 'CB_'.   Visual Basic's Combo boxes apparently are not the same as Windows Combo Boxes, experiment with the Combo Box examples to see which messages are relevant to a given Visual Basic Combo Box style.

The **Message Description** option of the **Example** pulldown is a reference only program describing the currently demonstrated messages, associated parameters and return values.

The message Value includes the constant WINUSER [h0400 = 1024].
        i.e. EM_LIMITTEXT = WINUSER + 21, we show the value equal to 1045.
The file WinMSG.Txt contains all of the message constants demonstrated in this release.
We will include all of the window message values in the next release.

Processing Messages

The Windows operating environment communicates with every application using event oriented messages.   Visual Basic supplies the majority of message monitoring required for our applications, although there are some very useful tools available in the windows message functions.   This release concentrates on some of the more immediately beneficial messages, we will expand this list in future releases.

Messages are sent to Windows via the SendMessage function supplied with a Message Value, and any additional information required.   Most messages are sent for a specific object and requires a Control Handle, which requires the use of the windows GetFocus() function.   View the section on Windows Control Handles.

Windows Control Handles

Windows messages can be directed to a window or an individual control.   The hWnd* parameter in the Windows Function 'SendMessage ' determines the object that receives the message.   The Visual Basic hWnd function will supply a Windows handle for a Form but not an individual control.   We therefore must use the Visual Basic SetFocus Method and the Windows GetFocus() function to obtain handles for individual controls.   View the SendMessage example for a demonstration of obtaining a Windows handle for a control.

SendMessage(chWnd* ,wMsg,wParam,lParam) as Integer

This function sends the message value (wMsg) to Windows (or a window).   The chWnd parameter identifies the window or object that receives the message.   Since Visual Basic does not readily supply a handle to an individual object we must use the windows GetFocus() function to obtain the chWnd value for control handles.

The additional parameters, wParam and lParam contain additional information relevant to the particular message sent.

Return: Dependent on the message sent, view the individual message descriptions.

Declaration   Example

GetFocus() as Integer

This function returns the Windows Handle for the control that currently owns the input focus. Use with the Visual Basic SetFocus Method to obtain the chWnd parameter value for the Windows SendMessage function.

Declaration Example

SetFocus Method
object.SetFocus: directs any user input or messages to the specified form or control.

hWnd is a reserved word in Visual Basic, therefore we used **chWnd** as the parameter name.

Message: CB_FINDSTRING Value=1036

Description: Finds first string in list box with given prefix string.

Remark: Apparently only works with Style #2 Combo Boxes.

wParam: Specifies where (index number) to begin searching rest of list.
lParam: Prefix string used for search.

Return: Index number of matching item or -1 (CB_ERR) if unsuccessful.

Message: CB_GETCURSEL   Value=1031

Description: Returns index of currently selected item in combo box list.

Remark: Apparently only works on Style #2, Combo Boxes.

wParam: Not Used.
lParam: Not Used.

Return: Index of selected item or -1 (CB_ERR) if no item is selected.

Message: CB_INSERTSTRING Value=1034

Description: Inserts string to the list box, ignores sorting.

Remark: Apparently only works on style #2, Combo Boxes

wParam: Index value for inserted string.
lParam: Insert string to be inserted.

Return: length of inserted string, -1 (CB_ERR)invalid index number, -2 (CB_ERRSPACE) insufficient space.

Message: CB_LIMITTEXT Value=1045

Description: Limits length of text allowed in Combo Box edit control.

Remark: CB_LIMITTEXT = 1025 but VB requires EM_LIMITTEXT value 1045.

wParam: Max number of bytes allowed.
lParam: Not Used.

Return: 1 successful,0 failed, -1 (CB_ERR) if non edit control available.

Message: CB_RESETCONTENT Value=1035

Description: Removes all strings from a Combo Box list.

Remark: Apparently works only on style #2, Combo Boxes.

wParam: Not Used.
lParam: Not Used.

Return: Not Used.

Message: CB_SELECTSTRING Value=1036

Description: Selects first string in list box that matches a prefix string. Identical to CB_FINDSTRING

Remark: Apparently will only work with Style #2, Combo Boxes.

wParam: Index of list, where search is to begin.
lParam: Prefix search string.

Return: Index of string if found. -1 (CB_ERR) if unsuccessful.

Message: CB_SETCURSEL   Value=1038

Description: Selects string in List box AND scrolls it into view.

Remark: Apparently works only on style #2, Combo Boxes.

wParam: Index of string to be selected, -1 sets list box to no selection.
lParam: Not Used.

Return: -1 (CB_ERR) if wParam is invalid.

Message: CB_SHOWDROPDOWN   Value=1039

Description: Force display of a Combo Box List.

Remark: Apparently works only on style #2, Combo boxes.

wParam: 1 (True) displays list, 0 (False) hides list.
lParam: Not Used.

Return: Not Used.

Message: EM_CANUNDO   Value=1046

Description: Determines if a edit control can respond to EM_UNDO.

wParam: Not Used.
lParam: Not Used.

Return: 0 CAN NOT respond to EM_UNDO, otherwise it can.

Message: EM_EMPTYUNDOBUFFER Value=1053

Description: Clears edit controls undo buffer, disables UNDO operations.

wParam: Not Used.
lParam: Not Used.

Return: Not Used.

Message: EM_GETMODIFY   Value=1032

Description: Used to determine if user has modified text in an edit control.

wParam: Not Used.
lParam: Not Used.

Return: Value of current modify flag.

Message: EM_LIMITTEXT   Value=1045

Description: Limits length of text user may enter in an edit control.

wParam: Max number of bytes allowed.
lParam: Not Used.

Return: Not Used.

Message: EM_UNDO Value=1047

Description: Undo last changes to an edit control.

wParam: Not Used.
lParam: Not Used.

Return: 0 if failed, otherwise successful.

Message: LB_FINDSTRING   Value=1040

Description: Finds first matching string in a list box for a given prefix string.

wParam: Index for first item to begin searching after.
lParam: Prefix string used in search.

Return: Index of located string if successful, -1 (LB_ERR) if not found.

Message: LB_GETCURSEL   Value=1033
Description: Returns index of currently selected item in List Box.

wParam: Not Used.
lParam: Not Used.

Return: Index of selected item, -1 (LB_ERR) if no item selected or multiple select list box.

Message: LB_GETSEL Value=1032

Description: Returns selection state of a given item in a List Box.

wParam: Index for item in List Box.
lParam: Not Used.

Return: 0 if not selected, >0 if selected, -1(LB_ERR) error in execution.

Message: LB_GETTOPINDEX Value=1039

Description: Returns Index number for first string visible in a List Box.

wParam: Not Used.
lParam: Not Used.

Return: Index of first visible item in List Box.

Message: LB_INSERTSTRING Value=1026

Description: Inserts string to List Box with out sorting.

wParam: Index position of new string, -1 if end of list.
lParam: String to be inserted.

Return: Index of insert position if successful, -1 (LB_ERR) if failed, -2 (LB_ERRSPACE) insufficient space.

Message: LB_RESETCONTENT Value=1029

Description: Removes all strings from a List Box.

wParam: Not Used.
lParam: Not Used.

Return: Not Used.

Message: LB_SELECTSTRING Value=1037

Description: Searches List Box for string matching a given prefix string.

wParam: Index where to begin search after in List Box, -1 top of list box.
lParam: Prefix string to use for search.

Return: Index of selected string, -1 (LB_ERR) if not located.

Message: LB_SETCURSEL   Value=1031

Description: Selects string in List Box AND scrolls it into view.

wParam: Index of string to be selected.
lParam: Not Used.

Return: -1 (LB_ERR) if not successful.

Message: LB_SETTOPINDEX Value=1048

Description: Set Index to first visible string in a List Box.

wParam: Index for first visible item.
lParam: Not Used.

Return: -1 (LB_ERR) if unsuccessful.

Message: WM_TIMECHANGE Value=1054

Description: Notify of change of system time, set hWnd to OxFFFF (65535) for all Top Level Win Apps.

wParam: Not Used.
lParam: Not Used.

Return: Not Used.

Message: WM_UNDO   Value=772

Description: Undo the last operation.

wParam: Not Used.
lParam: Not Used.

Return: Not Used.

Message: WM_WININICHANGE Value=26

Description: Notify of changes made to WIN.INI file. Set hWnd to 0xFFFF (65535) for all top level Win Apps.

wParam: Not Used.
lParam: String containing name of section modified, string DOES NOT have square brackets.

Return: Not Used.

```
'Declarations: MESSAGES

'Windows messaging requires that we use two API calls 'SendMessage to transmit
information to windows
'GetFocus to obtain Windows 'Handle' to a individual control
'Cannot use reserved hWnd as a parameter for SendMessage Function

Declare Function SendMessage Lib "User" (ByVal chWnd%, ByVal nMsg%, ByVal wParam%,
ByVal lParam$) As Long
Declare Function GetFocus Lib "User" () As Integer
```

```
Sub SendMessageButton_Click ()

        Source code is shipped with registered disks....
```

Wildcat

Software

WinDLL Fastrack: Programming Notes

For information on how to use help:
choose Help - Using Help.

Registering this disk:

Why should YOU register,

You get the **most current version** of this disk.(We have made improvements!)
You get the source code for the WinDLL programs.
All following updates are only $10.00
You are notified of changes to your disk and about new programmers tools.

Suggested registration price: $19

Wildcat Software
PO Box 2607
Cheyenne, Wyoming 82003
Attn: Windll Fastrack

We welcome any suggestions that will help improve this program, please feel free to write or contact us on CompuServe.   Our CompuServe Id is 76675,122.

The Window Dynamic Link Libraries

Visual Basic DLL declarations require that we state the Dynamic Link Library where the function is located.   There apparently are 4 Windows function libraries: **Kernel**, **User**, **System** and   the **GDI**.
If you wish to experiment with functions not covered in this release, try referencing one of those libraries.

Windows Data Types and Visual Basic Equivalents

The following table lists the Windows data type with respect to using Windows function calls. The **VB Parameter** list recommended types to use as a function parameter or return type. Use the **VB Structure** type in structures that the Windows DLL will access.

Windows	**VB Parameter**	**VB Structure**
BOOL	Integer (AND)	String * 1
BYTE	Integer (AND)	String * 1
char	Integer (AND)	String * 1
dWord	Long	Long
HANDLE	Integer	Integer
int	Integer	Integer
LONG	Long	Long
LPSTR	String ($)	String * *N*
short	Integer	Integer
void	non-TYPE*	- -
WORD	Integer (+)	Integer (+)

See also: Naming conventions; Microsoft Windows Programmers Reference.

Naming conventions: Microsoft Windows Programmers Reference.

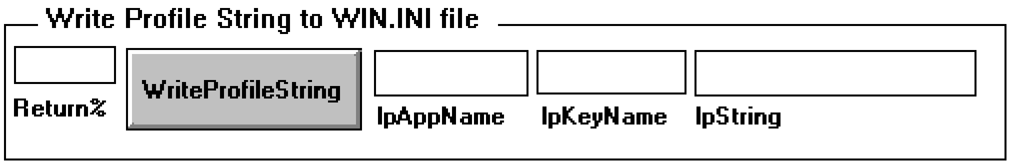
The naming conventions used for parameter names in the Microsoft Windows Programmers Reference were retained in the sample code regardless of data type conversions for Visual Basic variables.

Mircrosoft's parameter names use an italic prefix to indicate the parameters data type. Following is a list of Mircrosofts Prefixes, Data Types and resulting Visual Basics type.

Prefix	Type	Visual Basic Type	Example
b	BOOL	Integer	bStat%
c	BYTE	Integer	cDriveLetter%
c	char	Integer	cChar%
dw	LONG	Long	dwFlag
f	bit flags	Bitwise Character	String*1 or Integer
h	HANDLE	Integer	chWnd%
l	LONG	Long	lParam
lp	LongPointer	String ($)	lpAppName$
n	Short	Integer	nSize%
p	Short	Integer	pMsg
w	Short	Integer	wUnique%

Naming Conventions:
See Also: *Naming conventions used in Mircrosofts Windows Programmers Reference.*

The sample programs are oriented to give you a quick understanding of the Windows functions without forcing you to dissect elaborate program code. Most of the functions are designed to operate as separate entities, although they are assembled in groups where they can be used together. Each function is displayed as a named command button and associated parameter fields.



The sample above shows a typical function example. To test this example you would supply the field parameters **lpAppName**, **lpKeyName** and **lpString**. Clicking the **WriteProfileString** command button would execute the function with your supplied values. The source code for this function would be found in the subroutine **WriteProfileStringButton_Click().** The the controls containing the supplied parameters are named using the capitol letters of the function name followed by an underscore "_" and the parameter name. (**WPS_lpAppName, WPS_lpKeyName, WPS_lpString and WPS_Return**)

The subroutine, prior to calling the function, converts all the parameters to the proper data type, *using only local variables, except where data structures are used.*
ie:

```
lpAppName$ = WPS_lpAppName.Text
lpKeyName$ = WPS_lpKeyName.Text
lpString$        = WPS_lpString.Text

ret% = WriteProfileString(lpAppName$, lpKeyName$, lpString$)

WPS_Return.Text = Str$(ret%)
```

Of course, you find the sample code a little more complicated than the above example, but we kept it as simple as possible while trying to avoid execution errors.

Unrecoverable Application Errors

Making a Dynamic Link Library call removes us from Visual Basics safety blanket and errors can crash the Windows Operating Environment.   Save your program prior to testing it, or risk the AGONY OF DELETE.

While writing this code we caused Unrecoverable Application Errors in two ways.

FIRST METHOD: Using an undefined parameter in a function call.

Visual Basic does not require us to define variables prior to their being used.   This can be a problem if we begin to make calls outside the Visual Basic operating environment. If a Windows function returns a value to one of its parameters, we MUST create that parameter prior to calling the function.   If the parameter is a string BE SURE IT IS AT LEAST ONE CHARACTER IN LENGTH.   Windows does not like basic's null length strings. If the function requests the length of a parameter string, BE SURE THE STRING IS AT LEAST AS LONG AS YOU SAY IT IS.

SECOND METHOD: Not declaring a function return type.

This error caused a hour of confusion for us one day.   Every Windows function returns a value which is 'typed' in the function declaration.

```
i.e.    Declare Function GetFocus Lib "Kernel" ( ) as Integer
```

Not having the 'as Integer' type following the statement would have caused a runtime error,   if my program hadn't caused a Unrecoverable Application Error first.   This CRASH can be knarly to find because the 'as type' part of the declaration is usually not in view on the edit screen.

Working with Bitwise Data

A quick refresher course on bitwise operations.

Bit operations: Many of the Windows DLL's return values should be read as a Bit Flags. Listed below are <u>eight possible bit flags</u> and values.

Bit Position	Byte	Value	Basic Exponential
0	0000 0001	1	2^0
1	0000 0010	2	2^1
2	0000 0100	4	2^2
3	0000 1000	8	2^3
4	0001 0000	16	2^4
5	0010 0000	32	2^5
6	0100 0000	64	2^6
7	1000 0000	128	2^7

If more than one bit flag is set in the byte the value becomes the sum of the flag values.

Example: If Bit #1, Bit #5 and Bit #6 were set then
Byte is 0110 0010
Value is 2 + 32 + 64 = 98

The basic 'AND' operator allows us to test for Bit Flags.

Example: Testing Byte 0110 0010 = 98

Byte Value	AND	Test Value	= Result	Bit Flag Set
98	AND	1	= 0	No
98	AND	2	= 2	Yes
98	AND	4	= 0	No
98	AND	8	= 0	No
98	AND	16	= 0	No
98	AND	32	= 32	Yes
98	AND	64	= 64	Yes
98	AND	128	= 0	No

The basic exponential allows a fast bit map testing

Example:

```
  Program..
   ByteVal = 98
   For bit = 0 to 7
          If ByteVal AND 2^bit Then Print "Bit "; bit; " set."
   Next
 Prints...
   Bit 2 set.
   Bit 5 set.
   Bit 6 set.
```

Sending and Receiving the BYTE, BOOL & Char data types.

The C language BYTE, BOOL & Char data types are one byte variables not supported by Visual Basic, but there is a work around.   Sending BYTE data is quite easy since the you can pass any BYTE variable as an integer. (the smallest object that can be 'stacked' in the PC)

Receiving a BYTE result is a little more tricky.   Keep in mind that you are receiving an integer with only one byte of valid information.   We worked our way around this by using the 'And' operator with an integer equal to 255.

Example: From the GetTempDrive* function that returns a temporary drive letter as a BOOL.

```
Problem:   We expect a return   value range of 0 to 255 for a BOOL
           But instead we get a   the return value;    ret% = 14915

Solution:   tmpDrive% = ret% And 255         'AND' the return with 255
         ? tmpDrive%                                  'prints   "67"
         ? Chr$(tmpDrive%)                        ' prints   "C"
```

This example only deals with a single byte.   Integer bit flags are occasionally used by the windows routines.   You can expect to get a Long with them, also.

The GetTempDrive function is in WinDLL's example code project WIN_SYS.

AND the return value of this parameter with 255, see the section on ***BYTE, BOOL & Char data types***.

We prefer the BASIC **String$** type for parameters, remember to allocate sufficient space for the return string. If Windows writes to the variable, remember that the last character in the string will be a null.

For Structures we must use the VB **String * n** Type.   If you use String * n types for parameters remember that the last character in the string is a null.   Make **n** equal to the length of the longest expected return string, + 1, for the null character.

A **WORD** is an unsigned integer. If you operate on WORD variables, remember that negative integer values are greater that 32767. Passing a negative integer as a WORD is viewed as an unsigned integer by Windows.

A **Void** is a return only parameter.   Declaring a function without the **'AS *TYPE*'** is equivalent to a void Windows function.

hWnd is a reserved name in Visual Basic so we substituted **chWnd** (*c*ontrol *h*andle)

Visual Basic does not support bitflag operations see the section: ***Working with Bitwise Data.***